D2.3

Interim Version of the ELLIOT Platform

Deliverable data

Deliverable no & name	**D2.3 – Interim Version of the ELLIOT Platform**
Main Contributors	*POLY*
Other Contributors	*UNOTT, INRIA, UR, BIBA*
Deliverable Nature	*Prototype*
Dissemination level	**PU** Public **X** **PP** Restricted to other programme participants (including the Commission Services) **RE** Restricted to a group specified by the consortium (including the Commission Services) **CO** Confidential, only for members of the consortium (including the Commission Services)
Date	31/03/2012
Status	*Final*

Document history

Version	Date	Author /Reviewer	Description

Table of Contents

Table of Figures

List of Abbreviations

API	Application Programming Interface
BL	Business Logic
GUI	Graphic User Interface
GWT	Google Web Toolkit
JSR	Java Specification Requests
RDF	Resource Description Framework (RDF)
SSO	Single Sign-On
SPARQL	SPARQL Protocol and RDF Query Language
UML	Unified Modeling Language
UX	User Experience
WS	Web-Service
WSDL	Web Service Definition Language

1 Executive Summary

This document accompanies the delivery of the interim version of the ELLIOT Platform created in the scope of Task 2.3 of the ELLIOT project.

The starting point for the technical implementation is the feedback received on the initial concept demonstrator provided in the context of Task 2.2 (reported in D2.2) from the end-users and the other partners of the project, and the detailed application scenarios developed in parallel to technical work (D4.2.1 and D4.3.1). The technical prototype described in this document is made available on the web for further interaction with testbed users, again based on the Living Labs approach for another round of validation and verification activities.

The deliverable provides an overview of the deployment of the solution and reports some details about the major component of the platform developed by the different ELLIOT partners.

A user manual is part as well of this deliverable.

2 Introduction

2.1 Purpose, Intended Audience and Scope

The purpose of the document is to outline the delivered technical prototype created in the scope of Task 2.3 implementing the Interim Version of the ELLIOT Platform. The final version of the ELLIOT platform will be provided in M27 and reported in D2.4.

The intended audience of this deliverable is a technical audience, even if a user manual will be contextually released and provided to the real users of the platform coming from ELLIOT Living Labs.

2.2 Applicable Documents

List of applicable documents:

AD(1). ELLIOT Description of Work (DoW)

AD(2). ELLIOT D1.1 (KSB Experience Model Overall Framework)

AD(3). ELLIOT D1.2 (Knowledge Experience sub-model)

AD(4). ELLIOT D1.3 (Social Experience sub-model)

AD(5). ELLIOT D1.4 (Business Experience sub-model)

AD(6). ELLIOT D2.1.1 (User Requirements and Architectural Design – First Version)

AD(7). ELLIOT D2.2 (Fast Prototype)

AD(8). ELLIOT D3.1 (State of the Art for IOT oriented User Co-Creation)

AD(9). ELLIOT D3.2 (Report on Requirements Engineering)

AD(10). ELLIOT D3.3 Serious Gaming Approach to support Usability of IOT oriented User Co-Creation

AD(11). ELLIOT D4.1 (Specification of the IOT use-cases)

AD(12). ELLIOT D4.2.1 (Reports on IOT Living Labs Methodology and tools - initial)

AD(13). ELLIOT D4.3.1 (Report on IOT Living Labs Continuous Exploration and evaluation - initial)

ELLIOT D6.1 (Project Handbook and Quality Plan)

3 ELLIOT Platform Interim Prototype

This chapter will describe at high level, the v1.0 of the ELLIOT Platform; the section is composed by two major parts: in the next chapter 3.1 a short overview will be provided to the whole system and the deployment diagram will be outlined, in the following chapters the major components implemented will be described.

3.1 Overview

The solution implemented is multi-tenant platform composed by several modules; the human actors will interact with this complex system by the GUI provided by the core component of the platform and made available on the web. The suggested browser to manage the GUI is Firefox.

Version	**Prototype - v1.0**
Availability	**GUI web-address (changed from D2.2)** **http://demos.polymedia.it/elliot/ep/** **Guest credentials:** - **username:** *guest* - **password:** *guest* - **role:** *Living Lab Manager* **(Firefox suggested)**
Contact Person	**michele.sesana@polymedia.it**

Here below a short overview of the architecture is reported; for the details of the solution and the roles of different small components we suggest to the reader to read ELLIOT D2.1.1.

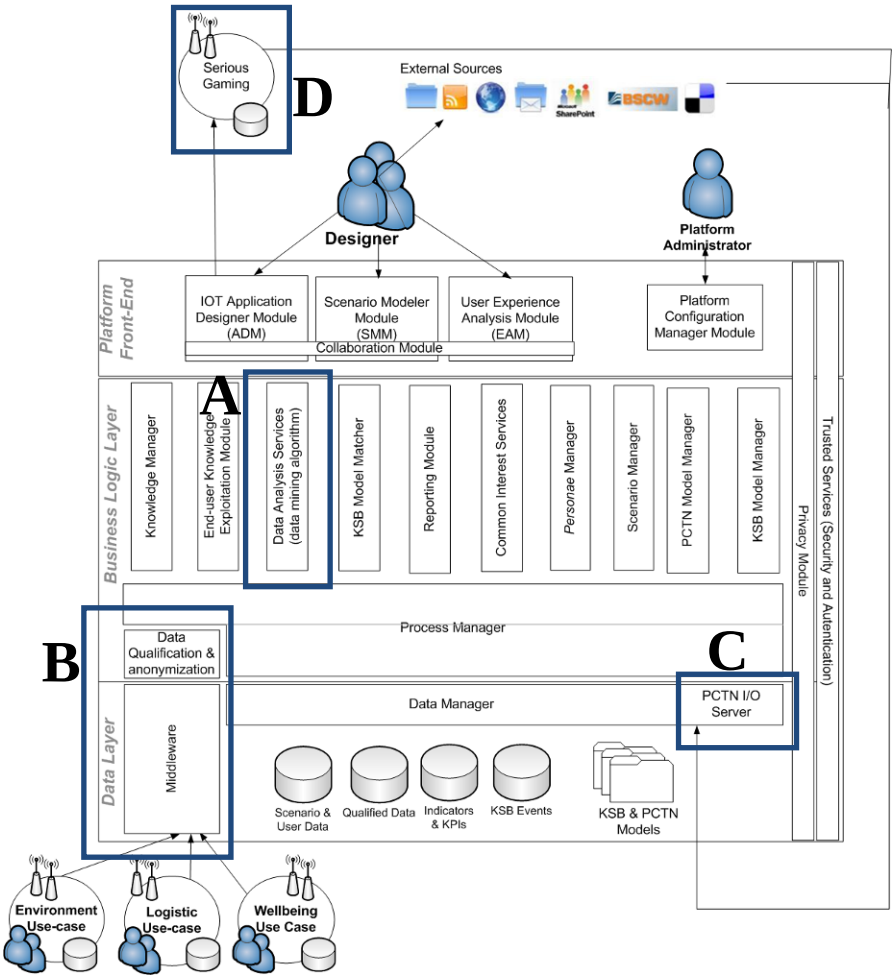


Figure 1: ELLIOT Platform Overview of the Architecture

To facilitate the reader in the comprehension of the solution the major components described in next chapters have been highlighted in the architecture, the remaining ones are part of the CORE part of the platform, divided in the three layers: presentation (GUI), services and data.

- chapter 3.2 describes the GUI of the application and in annex A the user manual is provided, as introduced before this is the component by which the user will interacts will all platform functions;
- the CORE platform reported in chapter 3.3 takes in charge of the provision of services necessary to manage the internal platform process and the connection with the other

components;

- 3.4 gives an overview of the platform data level where data is stored and provides the basic functions to manage and aggregate data. Disaster recovery is part of this component;
- the ELLIOT data mining component based on a FocusLab server is depicted in the picture with the *"A"* rectangle. Relying on this component the ELLIOT platform has the capability for offering to the user the possibility to manage big set of data and analyse them by data mining algorithms ;
- in the picture the component described as *"B"* is the ELLIOT Middleware; it is used for the data ingestion from the Living Labs and its anonymization and qualification, it includes also a thin client to be deployed on the Living Lab premises. It is based on Hydra (aka LinkSmart) and described in chapter 0;
- the ELLIOT PCTN component is highlighted by the "C" rectangle;
- with "D" is pointed the ELLIOT Serious Gaming component under development in WP3, for this reason will be not described in this deliverable but we suggest to the reader to read D3.3 to have more information about it.

In the next figure the deployment schema is reported. The core platform is installed in server 1 (POLY – Milano – Italy) and manages the connection among the different processes due to the fact that CORE Platform and the ELLIOT middleware are installed there and provides the process management. The ELLIOT Serious Gaming Component is installed in Server 3 (BIBA – Bremen – Germany); the ELLIOT PCTN server is deployed in server 3 (UNOTT – Nottingham – UK) and the ELLIOT Data Mining Component is deployed on server 2 (INRIA – Sophia Antipolis – France). The ELLIOT thin client is installed on the LL servers, in the scope of the project six servers will encapsulate this component, one for each Living Lab.

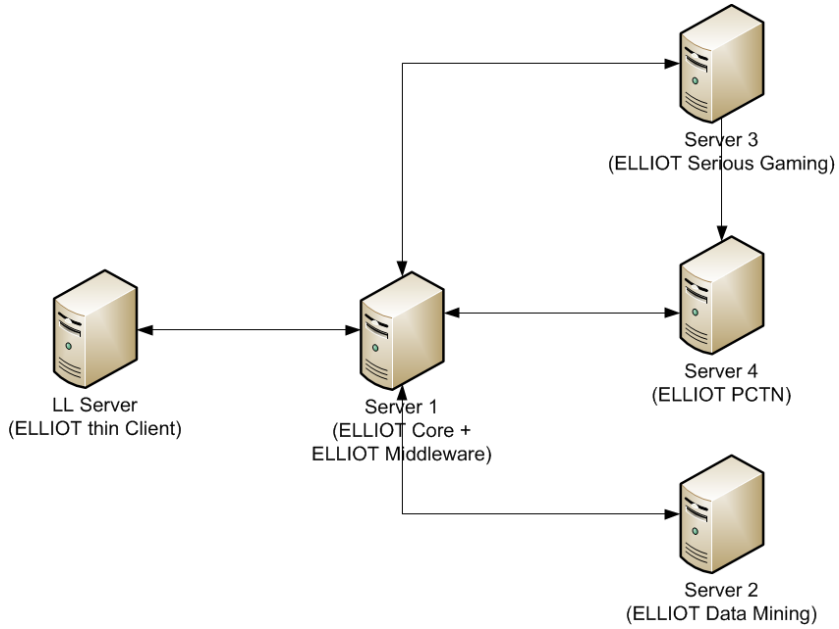


Figure 2: ELLIOT Experiential Platform Deployment Schema

As introduced before in next chapters the information about the ELLIOT major components will be provided.

3.2 Platform CORE (Presentation Layer)

The first version of the ELLIOT Platform CORE Graphical User Interface (GUI) is based on a web-application implemented in *Smart Google Web Toolkit (smart GWT)*[1] that allows the creation of appealing GUIs based on Ajax. To support a better visualisation of results jQuery and PHP have been applied where needed.

The development work focuses on the creation of the interfaces able to support the user in the Living Lab process (Co-Creation & Exploration, Experimentation and Evaluation). Along these steps the platform allows to instantiate and uses the KSB model analysing the results and understanding the user experience (UX).

3.2.1 Component Overview

Version	**Prototype - v1.0**
Availability	**GUI web-address (changed from D2.2)** **http://demos.polymedia.it/elliot/ep/** **Guest credentials:** - **username:** ***guest*** - **password:** ***guest*** - **role:** ***Living Lab Manager*** **(Firefox suggested)**
Host	**POLYMEDIA Server Milano - Italy**
Development	**Developed from scratch for the ELLIOT project**
Contact Person	**michele.sesana@polymedia.it**

3.2.2 Installation and Technical Requirements

The ELLIOT CORE package includes the three major components of the ELLIOT platform (presentation layer, service layer and data layer). The package contains two war files, one database script, and a non-SQL database.

Regarding the presentation layer the following steps should be done to install the component:

- The war file called "*ep.war"* should be deployed on the tomcat application server

[1] http://code.google.com/p/smartgwt/

running in a windows server. The war file contains all the libraries needed. The tomcat application server http port should be *"8080"*.

- Another war file called *"elliotgraphs"* should be deployed in an apache application server (port 80), the file contains all the libraries needed.

Host	**Server Windows (suggested: Windows Server 2008)**
Application typology	**webapplication**
Application Server	**Apache Tomcat 5.5 (http://tomcat.apache.org/download-55.cgi) – Apache 2.4.1 (http://httpd.apache.org/download.cgi)**
Major Technology	**Smart GWT**
Libraries	**GWT 2.2.0. Link http://code.google.com/webtoolkit/versions.html** **Smart-GWT 2.4. Link http://code.google.com/p/smartgwt/downloads/list** **LOG4j 1.2.16. Link http://logging.apache.org/log4j/1.2/download.html** **GWT-log 3.1.2. Link http://code.google.com/p/gwt-log/downloads** **jQuery 1.7.2** **http://code.jquery.com/jquery-1.7.2.min.js**
Development Environment	**Eclipse Helios. Link: http://www.eclipse.org/downloads/**

3.2.3 Component description

To show to the reader the GUI panels and functionalities a user manual has been provided in **Annex A**. This manual has been circulated to the end-users in order to support them at the time they are using the platform. The manual is described in a way to support a user willing to use the platform to co-create a service starting from the idea generation, passing through the monitoring of the real user activities and analyze the results, the KSB model span all over these phases.

3.3 Platform CORE (Service Layer)

The service layer is composed by several sub-components connecting the presentation layer to the data layer, managing the data process, providing function for disaster recovery, etc. In the following subchapters chapters 3.3.1 and 3.3.2 will show an overview and the installation procedure for the entire component; for the component description in chapter 3.3.3 a part of the platform has been selected to show the technical implementation of the services and the linked UML diagram have been reported and described.

3.3.1 Component Overview

Version	**Prototype - v1.0**
Availability	**POLY server in Milan** **Accessible by the GUI (chapter 3.2)**
Development	**Developed from scratch for the ELLIOT project**
Contact Person	**michele.sesana@polymedia.it**

3.3.2 Installation and Technical Requirements

The ELLIOT CORE package includes the three major components of the ELLIOT platform (presentation layer, service layer and data layer). The package contains two war files, one database script, and a non-SQL database.

Regarding the service layer the following steps should be done to install the component:

- the war file called "*ep.war"* should be deployed on the tomcat application server running in a windows server. The war file contains all the libraries needed. The tomcat application server http port should be *"8080"*;

Host	**Server Windows (suggested: Windows Server 2008)**
Application typology	**webapplication**
Application Server	**Apache Tomcat 5.5 (http://tomcat.apache.org/download-55.cgi)**
Major Technology	**Smart GWT**
Libraries	**GWT 2.2.0. Link**

	http://code.google.com/webtoolkit/versions.html **Smart-GWT 2.4. Link http://code.google.com/p/smartgwt/downloads/list** **LOG4j 1.2.16. Link http://logging.apache.org/log4j/1.2/download.html** **GWT-log 3.1.2. Link http://code.google.com/p/gwt-log/downloads**
Development Environment	**Eclipse Helios. Link: http://www.eclipse.org/downloads/**

3.3.3 Component Diagram

As introduced before inside the ELLIOT CORE data layer application a sub component has been selected to show in details the technical implementation occurred. The sub-components is devoted to the data gathering from the hydra component, the control of them and the storage in the two databases calling the data layer functions. In these diagrams information about the data layer component for completeness of the process, in the next chapter about the CORE Data layer the reader will find more information about the data structure involved will be provided.

Here below the following UML[2] diagrams are reported:

- Component diagram
- Class Diagram
- Detailed Class Diagram
- Activities Diagram

3.3.3.1 Component Diagram

Major components of the sub-component visible in next figure are:

- Hydra: registers itself on the Hydra Network; receives incoming data and stores them in XML format in a specific directory
- ElliotListener: routinely checks the inbound directory; filters new content; updates a dynamic listener system and a data structure containing updated information on current scenarios and their last update; unmarshal XML

[2] http://www.omg.org/spec/UML/

- ELLIOT Core Platform: creates and populates various data structures based on incoming XML files; stores data files; inserts different data in ER and RDF databases; manages queries; handles recovery mechanism
- ELLIOT jaxb classes: class representation of the XML structure exchanged between hydra and ELLIOTListener

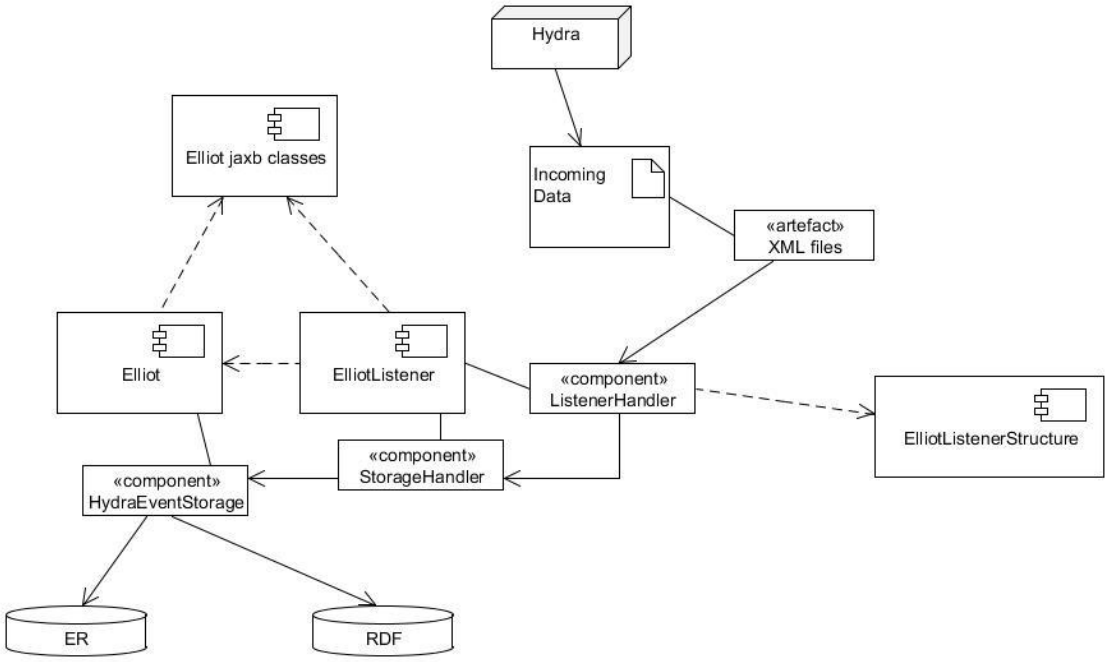


Figure 3: Component Diagram

3.3.3.2 Class Diagram

The class diagram shows the main classes involved in the process, in a single glance we can distinguish the module-based task handling:

- Hydra *DataProviderClient* class receives incoming data;
- *ListenerHandler* class manages all the filtering and listening-related work;
- *StorageHandler* performs checks and unmarshals the first file into a Record class; *HydraEventStorage* elaborates on the incoming Record, parses the second XML file, creates structures and sends orders to the *RDFManager* and *ERmng* classes. These last ones insert the aforementioned elaborated information into the appropriate Databases.

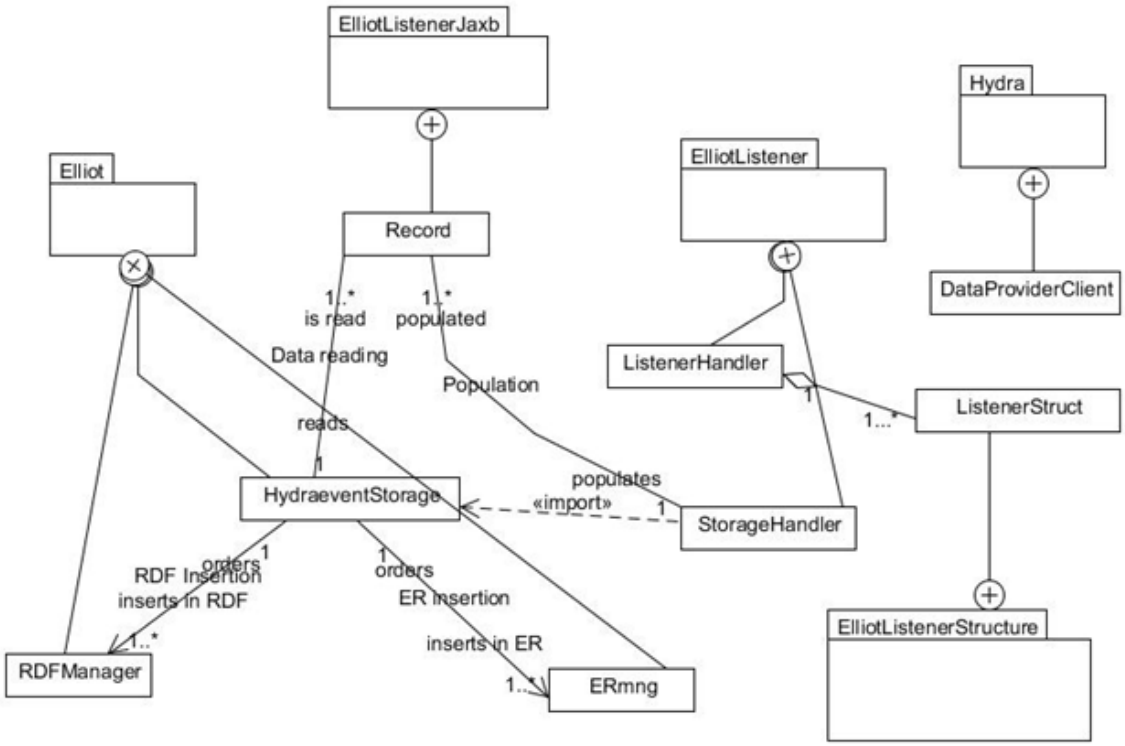


Figure 4: Class Diagram

3.3.3.3 Detailed Class Diagram

This diagram shows the details, in terms of methods and properties, of the component described in the previous chapter.

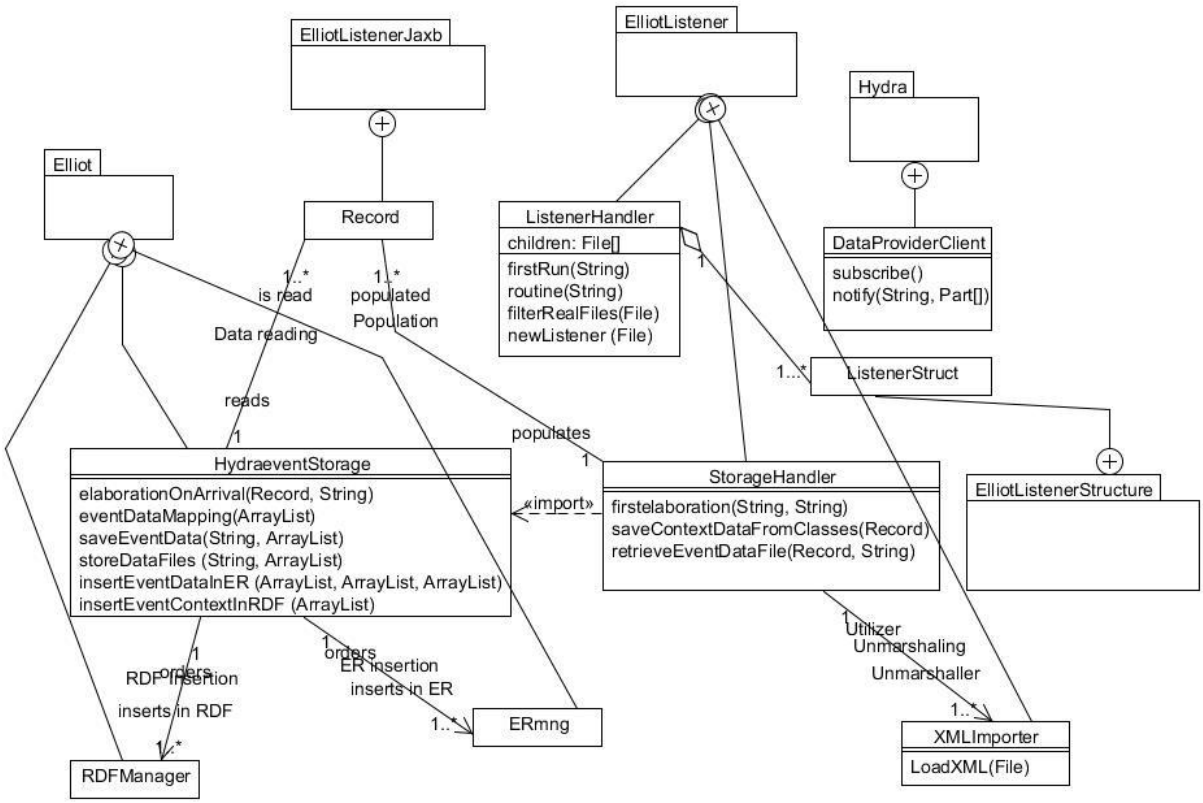


Figure 5: Detailed Class Diagram

3.3.3.4 Activity Diagrams

In this chapter three activity diagrams are described:

- Activity Diagram 1: LL data ingestion from Hydra (part 1)7
- Activity Diagram 2: LL data ingestion from Hydra (part 2)
- Activity Diagram 3: LL data disaster recovery

Activity Diagram 1: LL data ingestion from Hydra (part 1)

The first diagram describes the LL data ingestion from ELLIOT middleware (Hydra).

To accomplish this activity a listening application has been created. After the connection to Hydra waits for incoming data (push mechanism) and, at the time it receive something, stores received data into an XML files in the "inbound directory"

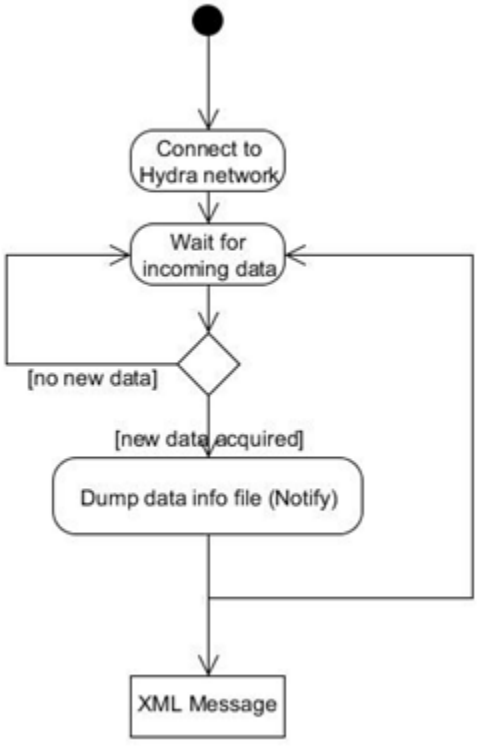


Figure 6: Activity Diagram - LL data ingestion from Hydra (part 1)

Activity Diagram 2: LL data ingestion from Hydra (part 2)

The listener routinely checks the "inbound directory"; applying various filters to the directory content to avoid file system related exceptions (directory exclusion, wrong file type exclusion), then apply several consistency checks controlling:

- if the file has been already processed (e.g.: duplicates);
- if the filename matches the standard pattern;
- if the file version is compatible with the existing specific single-scenario listening system

If one of those checks fails the file is automatically discarded.

If one or more files pass checks, then every file is processed (one by one). Those pertaining to an existing scenario update the meta-information present in the listening system. The completely new ones cause the creation of an *ad-hoc* specific listener. The first XML file is then *unmarshaled* and populates the built-in classes. This meta-data is used to populate the internal structures used to manage the process subsequent steps. The data meta-information are stored, the data files are retrieved and checked for consistency with the mentioned meta-information. If everything is correct and coherent, the second XML file is parsed, data is stored into internal structures, the file is moved and renamed for recovery purposes, then context data aresent to the data layer for storing into the RDF Database and into a dynamically created (if necessary) ER database table.

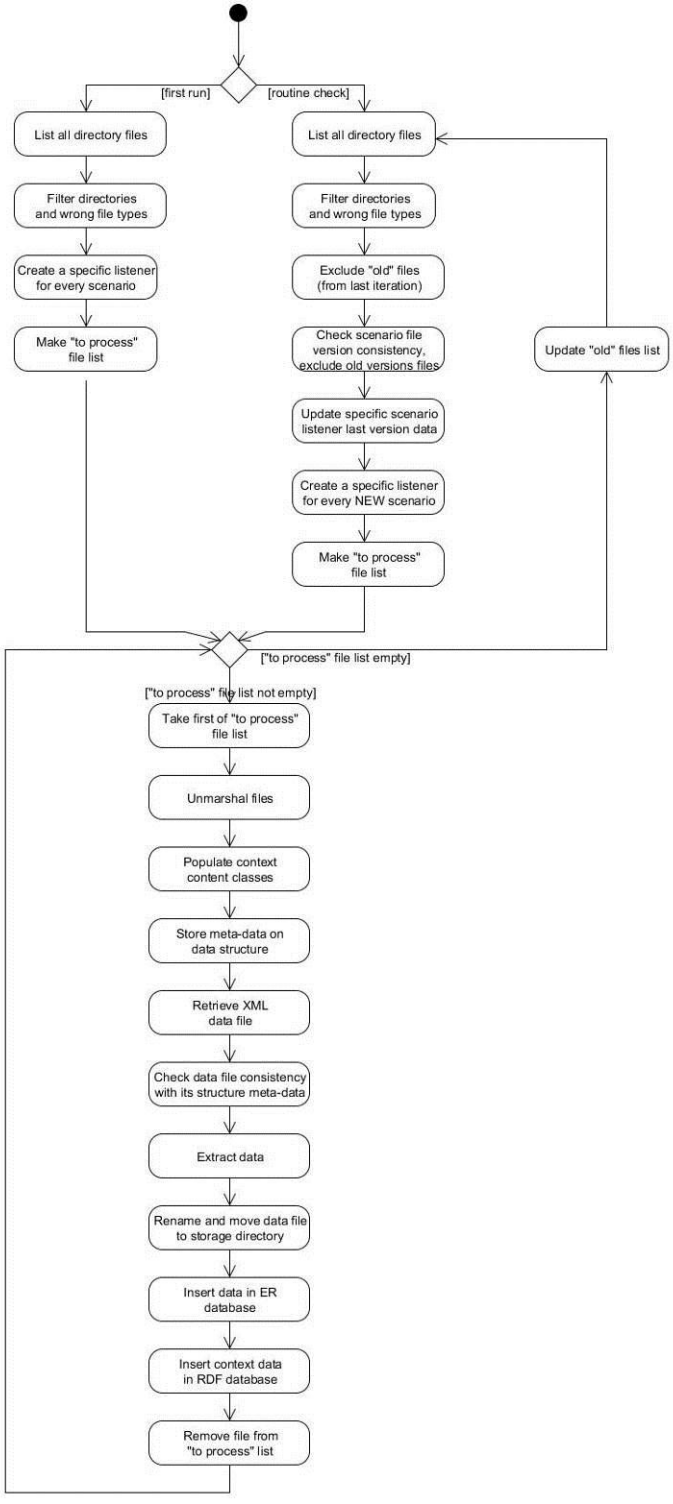


Figure 7: Activity Diagram 2- LL data ingestion from Hydra (part 2)

Activity Diagram 3: LL data disaster recovery

The data recovery system intervenes if the incoming query doesn't find the required information in the ER Database. Through the query a first set of relevant data is acquired. Then, a query is sent to the RDF Database to gather the additional information needed to find the correct data file. The file is recovered from storage and parsed, the data are loaded into the ER database and the original query is sent once again to the ER database.

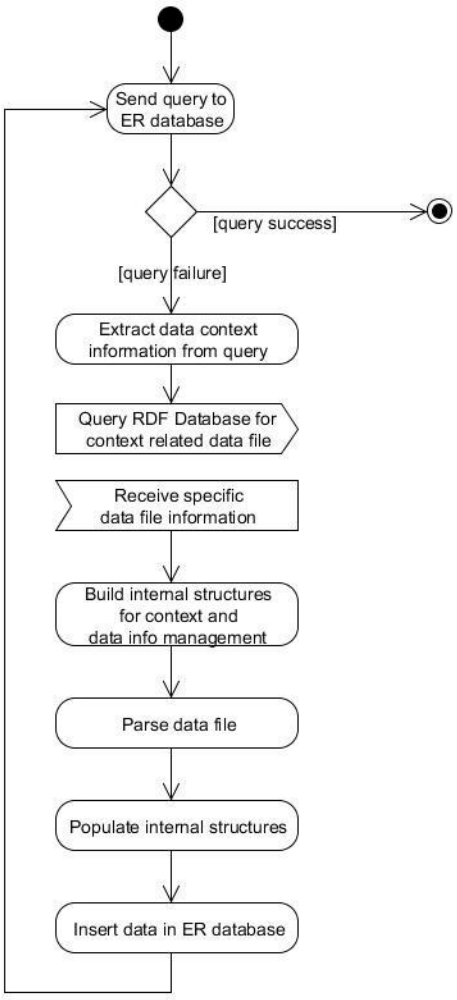


Figure 8: Activity Diagram 3: LL data disaster recovery

3.4 Platform CORE (Data Layer)

The data layer manages the data level of the application; it includes two major databases and a java application implementing the low level functions for handling data. The former item is a non-SQL database described in details in the fast prototype (D2.2) where all data of the application are stored, the latter is a SQL database where a subset of the data contained in the RDF structures saved in the non-SQL database are duplicated in SQL tables. The reason for the duplication is to increase the responsiveness of the system for specific activities that needs to use the SQL potentialities.

3.4.1 Component Overview

Version	**Prototype - v1.0**
Availability	**POLY server in Milan** **Accessible by the GUI (chapter 3.2)**
Development	**Developed from scratch for the ELLIOT project**
Contact Person	**michele.sesana@polymedia.it**

3.4.2 Installation and Technical Requirements

The ELLIOT CORE package includes the three major components of the ELLIOT platform (presentation layer, service layer and data layer). The package contains two war files, one database script, and a non-SQL database.

Regarding the data layer the following steps should be done to install the component:

- the war file called "*ep.war"* should be deployed on the tomcat application server running in a windows server. The war file contains all the libraries needed. The tomcat application server http port should be *"8080"*;
- at the time the ep.war file is deployed the filled called *"elliot.properties"* in the base directory of the application should be edited to point the right database inserting the machine IP and (if needed) new username, passwords and ports;

- the script *"elliotdb"* should be launched to install the database in postgres;
- to install the NON-SQL database:

Unzip *virtuoso-opensource.rar* provided in the installation package and containing already the ELLIOT graphs to the file system, for example C:

Add <yourDirectory>\virtuoso-opensource to system PATH

Execute from shell:

$ cd <yourDirectory>\virtuoso-opensource

$ start.bat

To check if the installation is complete the administration page (web frontend) is available at http://localhost:8890 (default username/password for administrator is dba/dba) and the list of ELLIOT graphs is visible at http://localhost:8890/sparql.

Host	**Windows Server 2008 Machine**
Application typology	**SQL database + non-SQL database + Java application**
Application Server	**Tomcat 5.5 (http://tomcat.apache.org/download-55.cgi)**
DBMS	**VIRTUOSO 6.1.3 (http://sourceforge.net/projects/virtuoso)** **Postgres 8.3 (http://www.postgresql.org/download/)**
Libraries	**SESAME 2.3.3 (http://www.openrdf.org/download.jsp)** **JDK6 (http://www.oracle.com/technetwork /java/javase/downloads/index.html)**
Development Environment	**Eclipse Helios. Link: http://www.eclipse.org/downloads/**

3.4.3 Component description

As introduced before the different components of the ELLIOT platform don't access in a direct way to databases and repositories; every request is mediated from the Data Manager java application. Figure 9 shows an overview of the data layer with the data manager on top and the non-SQL and SQL databases hidden by him.

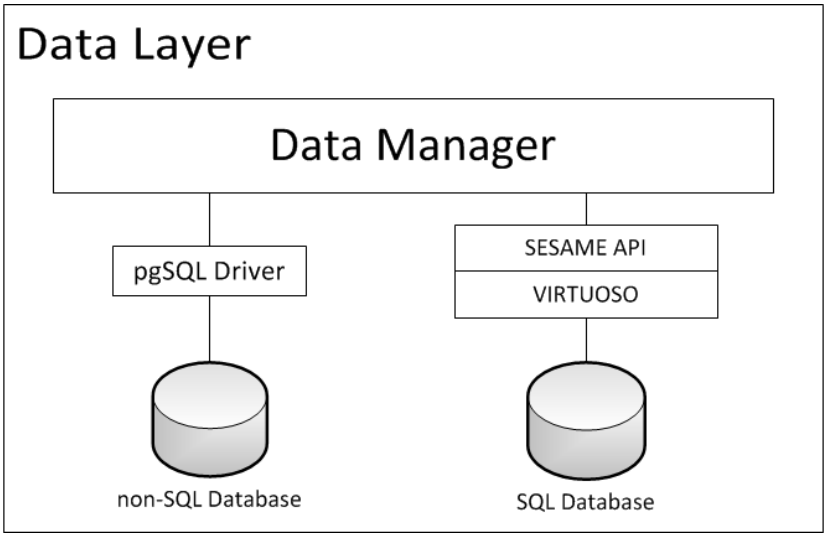


Figure 9: Data Layer Overview

The non-SQL database is implemented by an RDF structure described in D2.2, the structure is still valid but, in respect with the prototype, is used extensively while at that time few graphs were addressed at coding level. SESAME and VIRTUOSO are used for the management of the structure and the queries.

The SQL database is based on Postgres 8.3 and the connection managed by *pgDriver* realizing the management of connection and queries on it. While the RDF structure has been already introduced in D2.2 the SQL structure was not and so it is reported here below.

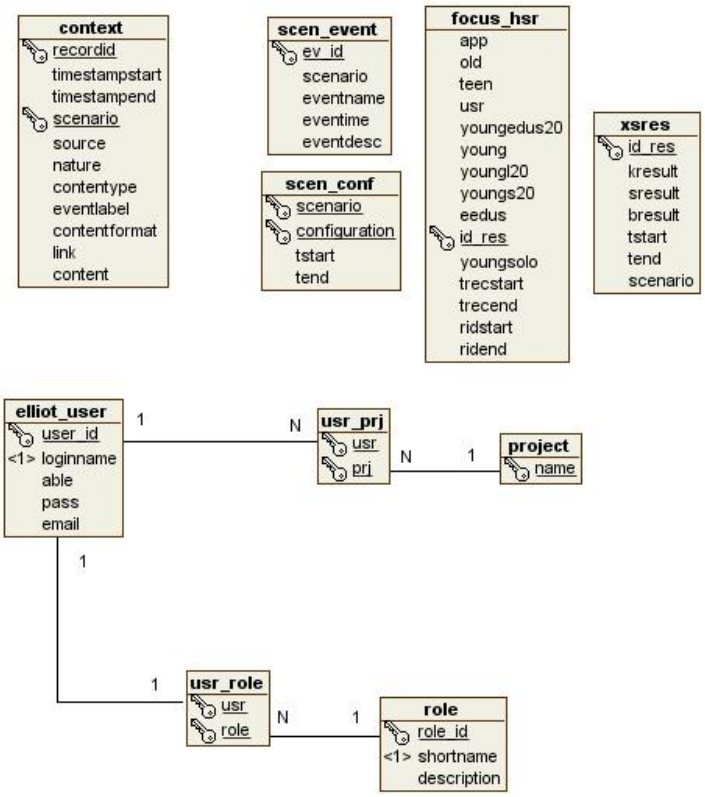


Figure 10:SQL-DB E-R diagram

The CONTEXT table stores all relevant information about incoming data batches: starting and ending times , the data stream nature (video, audio, log…) and original source (microphone, camera…), as well as detailed information about how data was gathered and (dynamic) content format and type. Finally it stores a link to the aforementioned data batch.

The SCEN_EVENT table stores information about both automatic and user-created events pertaining to every scenario: it contains the event title as well as the event timing and a detailed description of what happened.

The SCEN_CONF table stores scenario configurations and their start and end time, to allow a deep insight into the changes every scenario has undergone. Through such information the final user is able to optimize research into elaborated statistical information.

The FOCUS_HSR table stores the elaboration results from FocusLab Web Service call: dedicated to the HSR scenario, this table holds important statistical data about session timing and user age, as well as the session aim (educational) and applications used.

The XSRES table stores results of further elaboration: FocusLab Web Service data is selected in

a range of time and then passed through specific user-defined rules. This procedure generates different sets of Knowledge, Business and Social results. Everything is promptly stored in the database.

The ELLIOT_USER table stores users personal and login information, and is directly involved into the platform access security control

The ROLE table holds all roles users can be promoted to. Each of them has different prerogatives and access grants to different parts of the platform.

The PROJECT table stores scenario names as they are subsequently created, updated and deleted.

The USR_ROLE and USR_PRJ tables regulate and assure coherent user access to determined scenarios / platform functionalities.

Has to be remembered that the SQL databases is just a support of the RDF database for several operation requiring fast access to data like the export in a CSV file of a certain amount of events occurred in the Living Lab, so the structure of this database can easily change in the future to support other operations.

3.5 Platform Data Analysis Services

Data Analysis services provided by the Focuslab server (version 1.2) are used by the ELLIOT platform via web services.

FOCUSLAB v1.2 corresponds to the design and the implementation of a set of web-services providing basic and advanced functionalities for data analysis.

In this version, four web services are proposed:

- Linear regression.
- SCDS: we propose in this version to return as usage scenario the apparition frequency (min, max, average, slope) of a sequential pattern from data streams (SCDS algorithm) (see references [1,2])
- REGLO [1,3]
- Analyze HSR/ Analyze Log

These four web services are available for the ELLIOT platform. Other data analysis service might be added in the next versions of Focuslab from Living Labs needs.

A registry of services is provided to support the future connection of data there.

3.5.1 Component Overview

Version	**FocusLab V1.2**
Availability	**Inria server in Sophia Antipolis for FocusLab (for usage analysis algorithm)** http://www.ictusagelab.eu/focuslab/
Development	**Developed from FocusLab 1.0**
Contact Person	Brigitte.Trousse@inria.fr

3.5.2 Implementation and Technical Requirements

Host	**Linux Fedora FC14**
Application typology	**Web application**

Application Server	**Tomcat 7**
Major Technology	**Java6 and WebServices**
Libraries	**JDK**
Development Environment	**Eclipse. Link:** http://www.eclipse.org/downloads/

The services are installed on our tomcat server using a war deployment file (Focuslab.war).

Third party application can call services using http request (RESTful webservice implementation).

3.5.3 Component description

Focuslab (version 1.2) provides a set of services based on data mining algorithms that can be used for data analysis purposes in order to create measures or KPIs starting from structured data.

The web services provided by Focuslab takes as input structured data managed by the middleware through the data process and qualification module (the middleware is in charge of transforming raw data in structured data).

The web services of Focuslab will be executed from the process manager on collected data to extract indicators or measures from structured data (e.g: having in input a time series counting the number of people in front of a TV an algorithm can provide as output the tendency of the people in a certain time: ascending-descending-stable)

Data analysis services are registered on the services repository and accessed by the process manager ESB following SOA principles.

The process manager can consume the set of web-services in order to create KPIs starting from fine-grained data

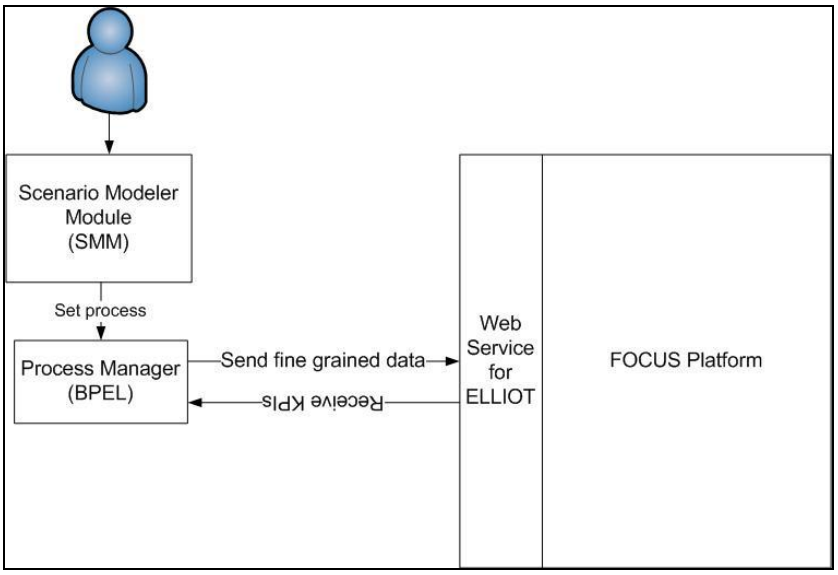
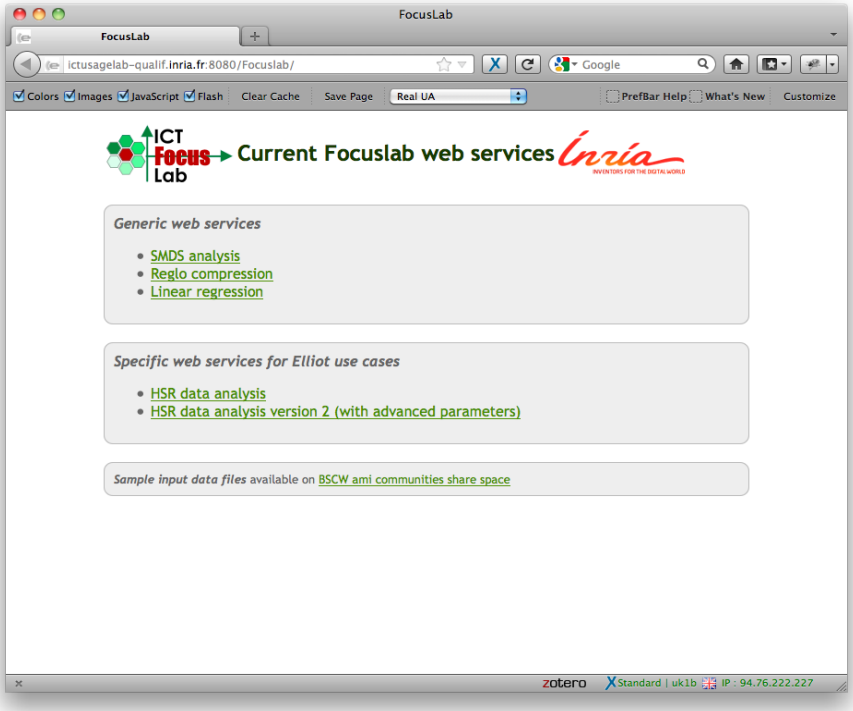


Figure 11: Data Analysis Services

The web services might be used by a web form or by http request in programming mode.

3.5.3.1 The web form access

The web forms can be reach on http://www.ictusagelab.eu/focuslab/, it offer an access to forms, which call services.



Each link in this page open a form to set parameters to named services. You would find parameters in the next section "API description".

3.5.3.2 API description

Each algorithm present in the Focuslab platform is available by calling method implemented as a jsp page.

In the following table you will find for each algorithm, the jsp to be call and the parameters.

Linear regression	regression_process.jsp	2 parameters: - A Key to identify the caller - A CSV file containing the input data to analyse e.g.: <value1>;<value2>
SCDS	analyseSCDS_process.jsp	4 parameters: - A Key to identify the caller - A specific sequential pattern we are looking for e.g. *game1; application 2* - a CSV data file containing the input data to analyze e.g.: <user_id>;<timestamp>;<event> - a CSV taxonomy file containing the taxonomy of the input data to analyze e.g.: <event>;<event_type_1>;<event_type_2>; This file is optional and could be used for pre-processing the CSV data file (replacing event by its <event_type>); For the moment we use only <event_type_1> as the high level type of the "event2e (item, visited page,

		used application, user action)
REGLO/GEAR	analyzeReglo_process.jsp	6 parameters: - A Key to identify the caller - A CSV file containing the input data to analyse (at least 1 column) e.g.: <value1>;<value2>;… - A memory size e.g.:40 - The method “Regression” or “Middle” - The output format “Html”, “XML” or “JPG” - The name of the columns of the CVS file
AnalyzeHSR	analyzeHSR_process.jsp	3 parameters: - A Key to identify the caller - A CSV file containing the input data to analyse e.g.: <value1>;<value2> - The output format “Html” or “XML”
AnalyzeLog	analyzeLog_process.jsp	6 parameters: - A Key to identify the caller - A CSV file containing the input data to analyse e.g.: <value1>;<value2> - The field number to be considered - A taxonomy file - The sequence to look for e.g.: game;mathematics - The output format “Html” or “XML”

For example, in Java, these web services could be called using the following template:

```
stastatic String servletUrl   = "http://www.ictusagelab.eu/focuslab/<serviceName>_process.jsp";
static String key=”XXXXXXXXXXXXX”;
static String dataFile     = "data.csv";
       HttpClient httpclient = new DefaultHttpClient();
       HttpResponse response = null;
       try {
          HttpPost httppost = new HttpPost(servletUrl);
          MultipartEntity reqEntity = new MultipartEntity();
          reqEntity.addPart( "key",   new StringBody(key));
          reqEntity.addPart("param1",  new FileBody(new File(dataFile)));
          reqEntity.addPart("param2",   new StringBody(memory));
          … … …
          httppost.setEntity(reqEntity);
          response = httpclient.execute(httppost);
      } catch (Exception e){                  }
      finally {
          try { httpclient.getConnectionManager().shutdown(); } catch (Exception ignore) {}
       }
```

3.5.3.3 Examples

We give in this section examples to call web services in Java.

- The first example calls HSR web service.

This web service returns information about usage of application from HSR logs file.

The input file look like:

```
31,26411,kanagram,elliot,2,2011-09-13 18:49:15
32,26411,kanagram,elliot,2,2011-09-13 18:49:25
33,8977,firefox-bin,elliot,1,2011-09-13 18:49:35
34,8977,firefox-bin,elliot,1,2011-09-13 18:49:45
```

It is a coma separated cvs file
(<id>,<process_id>,<application_name>,<user_name>,<face_count>,<date>)

Calling this service in Java could be done using the following code:

```
static String servletUrl   = "http://ictusagelab-qualif.inria.fr:8080/Focuslab/analyzeHSR_process.jsp";
static String dataFile     = "data.csv";
static String outputFormat= "xml";
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse response = null;
        try {
            HttpPost httppost = new HttpPost(servletUrl);
            MultipartEntity reqEntity = new MultipartEntity();
            reqEntity.addPart("csvdata",  new FileBody(new File(dataFile)));
            reqEntity.addPart("output",   new StringBody(outputFormat));
            httppost.setEntity(reqEntity);
            response = httpclient.execute(httppost);
       } catch (Exception e){                    }
       finally {
            try { httpclient.getConnectionManager().shutdown(); } catch (Exception ignore) {}
        }
```

Then the http response is an XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
 <APP_SESSION_NUMBER>308.0</APP_SESSION_NUMBER>
 <OLDERS_SESSION_NUMBER>73.0</OLDERS_SESSION_NUMBER>
 <TEENAGERS_SESSION_NUMBER>113.0</TEENAGERS_SESSION_NUMBER>
<USER_SESSION_NUMBER>308.0</USER_SESSION_NUMBER>
<YOUNGERS_EDU_SESSION_NUMBER_SHORTER_20S>0.0</YOUNGERS_EDU_SESSION_NUMBER_SHORTER_4
0S>
<YOUNGERS_SESSION_NUMBER>122.0</YOUNGERS_SESSION_NUMBER>
<YOUNGERS_SESSION_NUMBER_LONGER_20S>122.0</YOUNGERS_SESSION_NUMBER_LONGER_20S>
<YOUNGERS_SESSION_NUMBER_SHORTER_20S>0.0</YOUNGERS_SESSION_NUMBER_SHORTER_20S>
</xml>
```

This output gives requested informations about users using HSR interactive TV.

- The second example calls reglo web service with data taken from biba.

The input file biba.cvs is an extract of a database from BIBA which contain

```
12 29
-1 29
-4 29
12 29
4 29
. .
```

The two columns are:

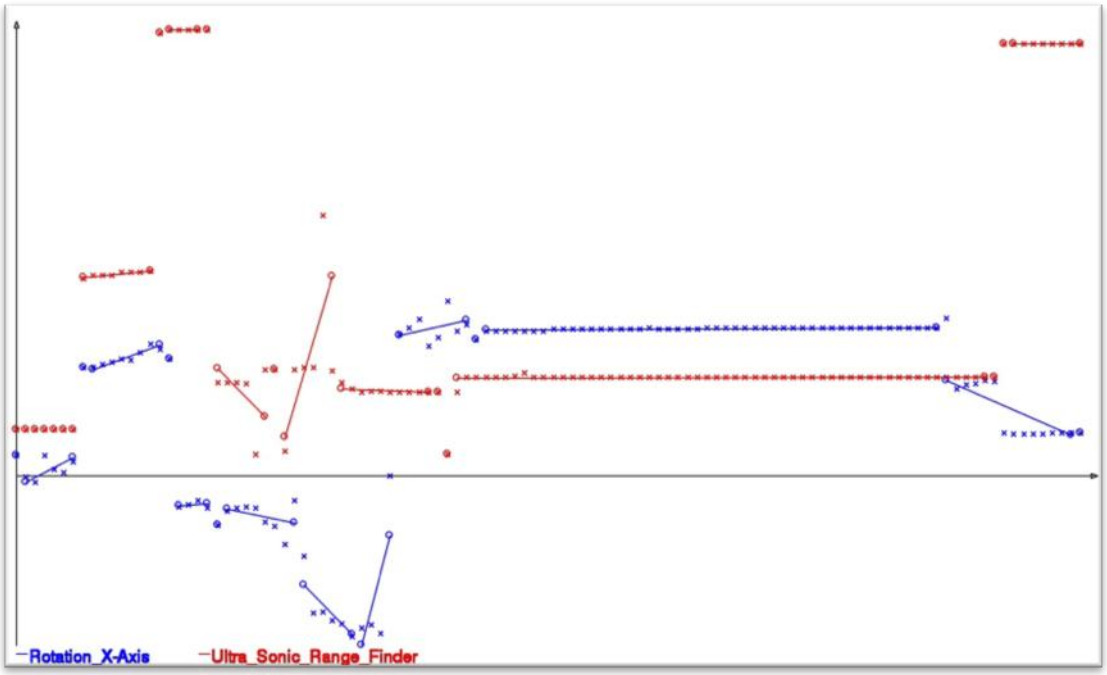
Sensor Rotation X-axis

Sensor Ultra sonic range finder

A sample of Java code that call the web service reglo_process.jsp:

```
static String servletUrl   = "http://www.ictusagelab.eu/focuslab/reglo_process.jsp";
static String key=”33e4a7975ed7ed16”;
static String dataFile     = "biba.csv";
static String memory= "40";
static String outputFormat= "jpeg";
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse response = null;
        try {
            HttpPost httppost = new HttpPost(servletUrl);
            MultipartEntity reqEntity = new MultipartEntity();
            reqEntity.addPart("csvdata",  new FileBody(new File(dataFile)));
            reqEntity.addPart( "memory",   new StringBody(memory));
            reqEntity.addPart( "key",   new StringBody(key));
            reqEntity.addPart("output",   new StringBody(outputFormat));
            httppost.setEntity(reqEntity);
            response = httpclient.execute(httppost);
       } catch (Exception e){                    }
       finally {
            try { httpclient.getConnectionManager().shutdown(); } catch (Exception ignore) {}
        }
```

Then the result is the following jpeg image file:



This kind of image might be used in co-creation workshop.

For the same data, the result can be given as a XML output looking like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xml>
   <compression>
      <variable name="Rotation_X-Axis" order="0">
         <point>
            <ts>1.0</ts>
            <value>12.0</value>
         </point>
         <segment>
            <point order="1">
               <ts>2.0</ts>
               <value>-4.0</value>
            </point>
            <point order="2">
               <ts>7.0</ts>
               <value>11.0</value>
            </point>
         </segment>
         <point>
            <ts>8.0</ts>
            <value>66.0</value>
         </point>
… … …
          </point>
      </variable>
      <variable name="Ultra_Sonic_Range_Finder" order="1">
         <point>
            <ts>1.0</ts>
            <value>29.0</value>
         </point>
… … …
          </point>
         </segment>
      </variable>
   </compression>
</xml>
```

Additional information on REGLO:

REGLO is an implementation of the history management strategy proposed in Marascu's thesis [1]. It takes a set of time series and provides a memory representation of these series based on a new principle, where salient events are important (in contrast to the recent events of decaying models).

Additional information on SMDS/SCDS:

SCDS (Sequence Clustering in Data Stream) is a clustering algorithm for mining sequential patterns (Java) in data streams developed by A. Marascu during her thesis. This software takes batches of data in the format "Client-Date-Item" and provides clusters of sequences and their centroids in the form of an approximate sequential pattern calculated with an alignment technique.

References:

[1] Alice Marascu. Extraction de motifs séquentiels dans les flux de données (in french), University of Nice Sophia Antipolis, 2009, Ph. D. Thesis, 2009

[2] Alice Marascu A, Florent Masseglia. Mining Sequential Patterns from Data Streams a centroid approach. In Journal of Intelligent Information Systems (JIIS), November 2006, vol 27, n_3, pages 291-307

[3] A.-M. Marascu, F. Masseglia, Y. Lechevallier. *A fast approximation strategy for summarizing a set of streaming time series*, in: *Proceedings of the 2010 ACM Symposium on Applied Computing, SAC'2010, Sierre, Switzerland*, March 22-26 2010, p. 1617-1621.

3.6 ELLIOT Platform Middleware

The ELLIOT middleware is the bridge between Living Lab and ELLIOT. It is based on the major outcome of the Hydra project: the Hydra middleware. The name of the open source middleware software is **LinkSmart** while the title of the project is has been developed in is "Hydra"; to remain coherent with previous deliverables, the middleware will be identified as "Hydra middleware" in this deliverable.

3.6.1 Component Overview

Version	**Executable version of Hydra middleware customized for ELLIOT**
Availability	**POL server in Milan for ELLIOT side and respective Living Labs**
Development	**Developed from the last available version delivered in Hydra**
Contact Person	**atta.badii@reading.ac.uk; m.tiemann@reading.ac.uk**

3.6.2 Installation and Technical Requirements

To install Hydra , the provided folder that contains the Hydra executable needs to be copied to the desired machine. That folder contains an executable file for starting the middleware as well as the necessary plug-ins and configuration files.

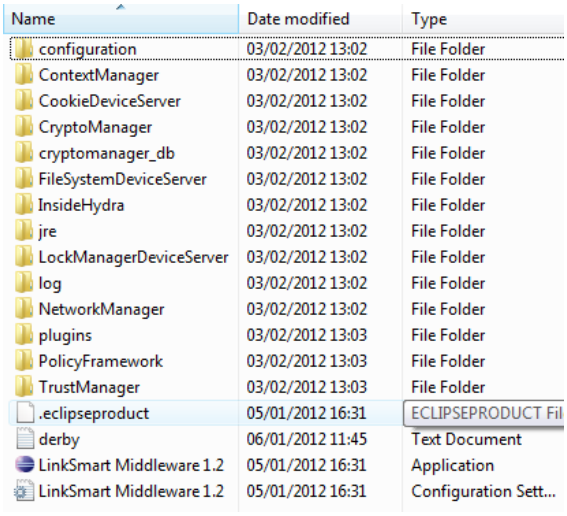


Figure 12: Contents of the ELLIOT Platform Hydra Middleware folder

Running this executable file starts a new instance of Hydra. To verify that Hydra is running successfully, open a browser and open the URL "http://localhost:8082/LinkSmartStatus" in a

web browser.

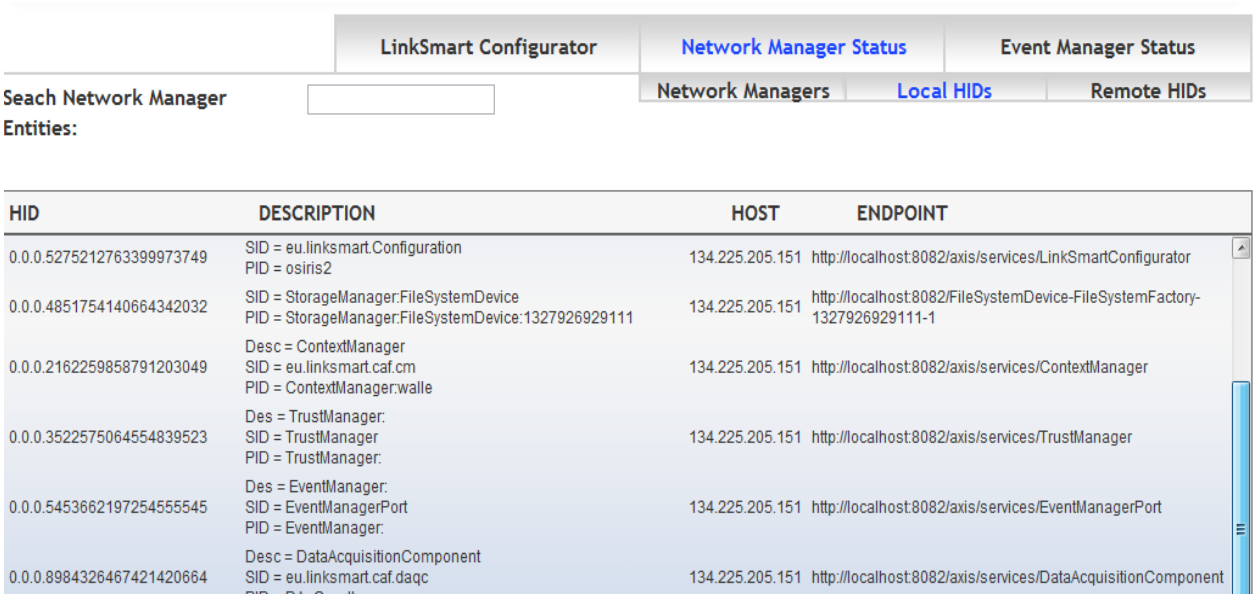


Figure 13: LinkSmart status page

Figure 13 shows the status page of Hydra when it is running successfully. Verify if all the internal managers of Hydra such as Network manager, Event manager, Context manager and Trust manager are displayed as web services under the Network Manager Status tab.

A Hydra ID or HID is a unique ID that identifies each instance of Hydra. Once an instance of the Hydra middleware is started, the default HID of the Network Manager is set to 10.10.10.10. Change this to a unique value that identifies your Network Manager. This can be done on the status page by clicking on the "LinkSmart configurator" tab and then changing the HID under eu.linksmart.network configurations as shown below.

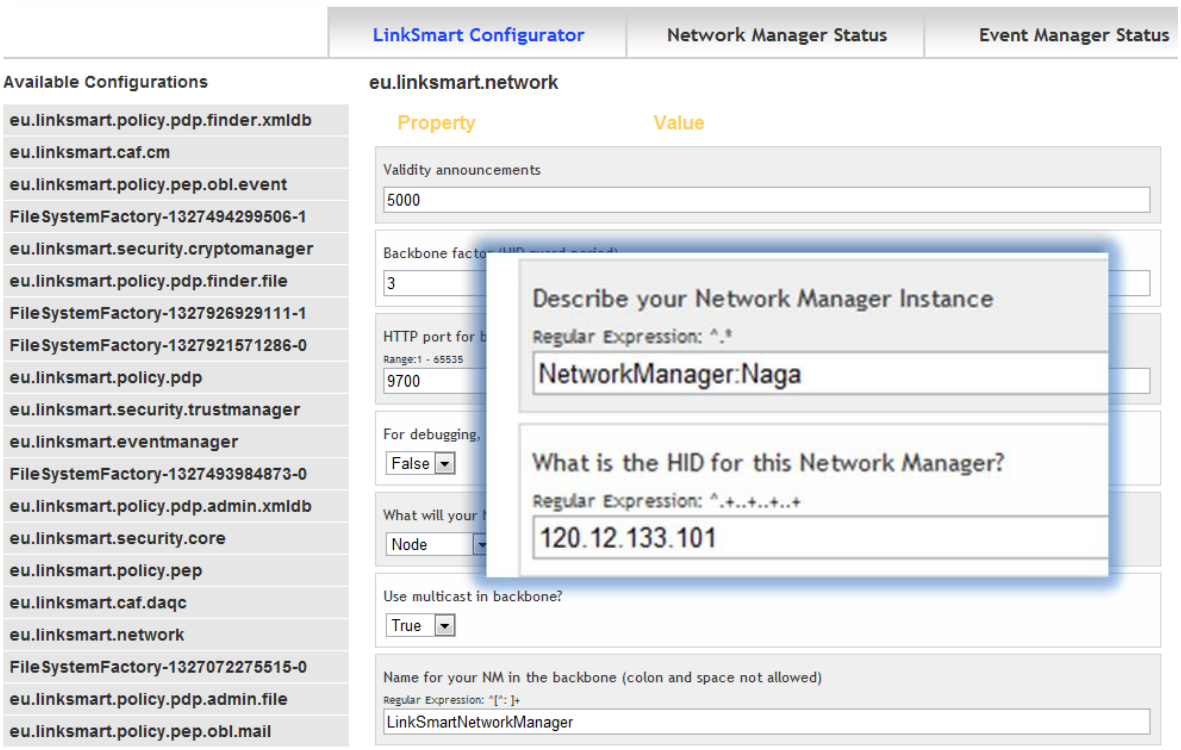


Figure 14: Editing Network Manager HID and description

Once the HID is changed, scroll to the bottom of the configuration web page and click on "Update configurations". To verify that the change of HID has taken effect, click on the "Local HIDs" tab under the Network manager status page. Check if the HID of the Network Manager reflects the modified HID.

When you have changed the HID of your local instance, the installation and verification process for the Elliot Hydra Middleware Platform is complete.

If the middleware is to be installed in a Living Lab, an XML file that specifies the configuration for data extraction needs to be placed along with the executable. Details of this specification are

provided in the next section.

Host	**Windows / Linux Machine**
Application typology	**Java OSGi framework application + IDE + knowledge base**
Application Server	**Java 1.6 or later**
Major Technology	**OSGi + Webservices**
Libraries	**JXTA, OSGi, UPnP, Apache AXIS, SOAP, XACML, DROOLS.**
Development Environment	**Eclipse Galileo. Link: http://www.eclipse.org/downloads/**

3.6.3 Component description

Hydra provides an intelligent middleware for networked embedded systems, deployable on both new and existing networks of distributed wireless and wired devices and capable of seamlessly integrating Internet of Things (IoT)-enabled devices into and within an Internet of Services (IoS) environment of Hydra Devices and Hydra Applications. In this context, the Hydra Middleware hides the underlying complexity of integrating devices and services into an Internet of Services and Things (IoTS) and supports solution integrators, device developers and application developers with development tools that enable the easy and cost-effective development and deployment of advanced IoT, IoS and IoTS systems. The middleware employs a Service-Oriented Architecture (SoA), to which the underlying communication layer is transparent. Hydra supports both distributed and centralised architectures and provides mechanisms to ensure security and trust, to support reflective properties and to enable the model-driven development of applications. The core middleware architecture consists of components that handle device discovery, secure networking, secure resource access and context management. Context-awareness is supported via a hybrid approach that provides capabilities for both powerful high-level reasoning, an advanced reasoning engine, and relatively lower-level semantic processing, enabled by the modelling of contextual data using an object-oriented key-value model.

The ELLIOT Platform middleware extends the basic Hydra middleware with a new application/plug-in that extends Hydra functionalities with the necessary ability to extract data from different Living Lab data sources such as databases, web services and log files. The ELLIOT platform middleware can buffer individual events or sequences of events (i.e. working

sessions) in a quasi-real-time fashion, provides mechanisms for anonymising extracted data and defines methods for data processing and analysis. The ELLIOT Platform Middleware component for the Living Lab side of the platform requires an XML configuration that specifies what needs to be extracted, where to extract the data from and how to anonymise it. In addition, rules for simple inference can be provided by Living Lab users. An interim version of the XML Schema that defines this XML configuration is provided in Annex B of this document.

Figure 15 provides an overview of the ELLIOT Platform Middleware.

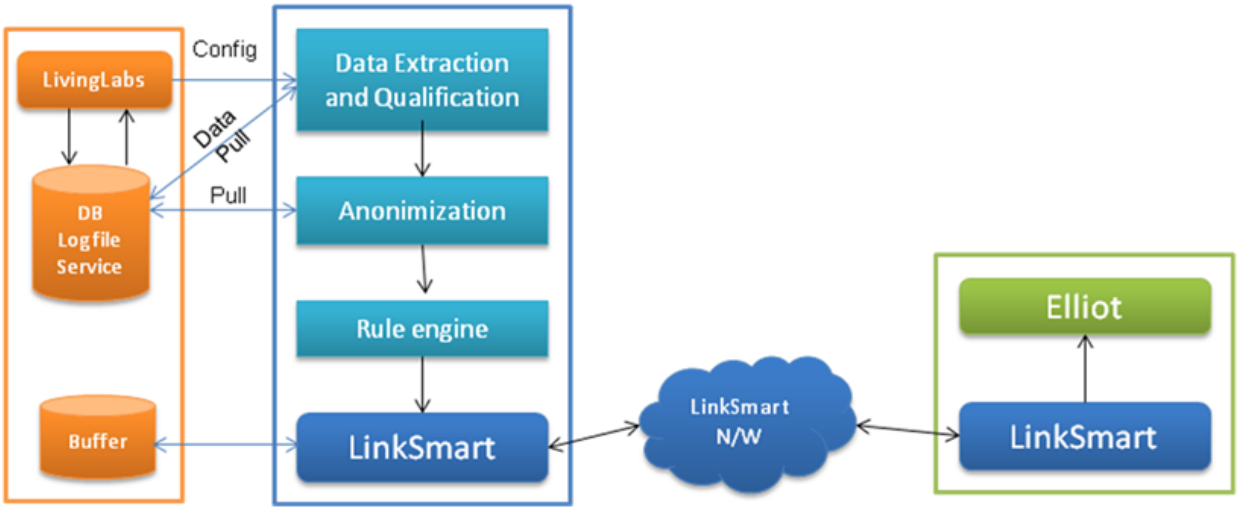


Figure 15: ELLIOT Platform middlware overview

The left side of the above figure depicts the Living Lab side of the middleware components described in this section. The Hydra/LinkSmart network (N/W) and the right side of the figure illustrate the connection to the ELLIOT platform.

The application also provides a Web Service method via which the ELLIOT platform can subscribe to the data extracted from a Living Lab; a call to this method adds the Elliot platform as a subscriber.

When subscribing, a notification frequency needs to be provided to the subscribe method call. The collected data-read-events will then be sent to the subscribing ELLIOT platform at the specified rate. Figure 16 shows the content of such an event in the XML format used.

```
<?xml version="1.0"?>
<Record id="1">
      <Header>
            <FirstTimeStamp>2012-01-30 16:00:25.0</FirstTimeStamp>
            <LastTimeStamp>2012-01-30 16:30:25.0</LastTimeStamp>
            <Scenario>hsr1</Scenario>
            <Content-type>log</Content-type>
            <Nature>log</Nature>
            <Source>log</Source>
      </Header>
      <Data>
            <EventLabel>read</EventLabel>
            <BufferId>hsr1_1328028106325_1</BufferId>
      </Data>
</Record>
```

Figure 16: An example read event

The "BufferId" that is contained in each event is the unique identifier that can be used to retrieve the actual data extracted from the Living Lab and stored in the ELLIOT Platform Middleware. The actual data can be retrieved via several Web Service getter methods using this identifier; stored data can also be deleted by referencing the buffer identifier.

The actual data collected from Living Lab data sources are structured in XML format. Figure 17 depicts an example data record in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Records>
      <Record>
            <timestamp>2012-01-30 16:33:25.0</timestamp>
            <face_count>1</face_count>
            <process_name>play</process_name>
            <constant1>somevalue1</constant1>
      </Record>
      <Record>
            <timestamp>2012-01-30 16:33:26.0</timestamp>
            <face_count>2</face_count>
            <process_name>play</process_name>
            <constant1>somevalue1</constant1>
            <multiple>true</multiple>
      </Record>
....
....
</Records>
```

Figure 17: An example actual data record

The first version of the ELLIOT hydra-based middleware and the interim version of the protocol have been provided to the end-users in this fast prototype to elicit further comments and to allow a preliminary check in terms of suitability to WP4 Living Labs developments.

3.7 ELLIOT Platform PCTN

People-Concepts-Things Networking (PCTN) for ELLIOT is an extension of People-Concepts Networking (PCN) that was developed during the Laboranova EU research project. PCN main goal is to provide "knowledge connection" among people through the use of their most relevant concepts (could be compared to spreading pheromones) that appear in the information they produce (e.g. documents, blog entries, web pages). These concepts, also named tags, form a kind of user's profile. They also act as links to retrieve users' content objects.

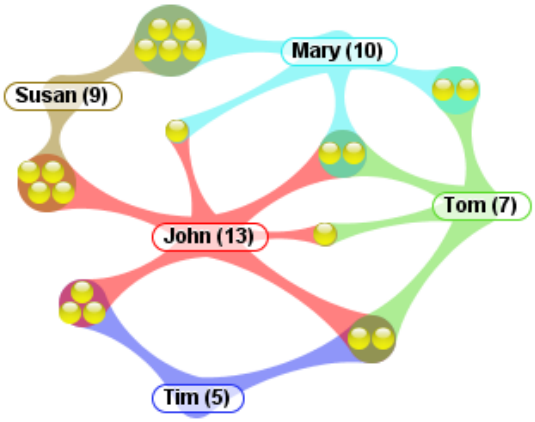


Figure 18: Users are connected via shared concepts (tags) linking to content assets (knowledge resources)

PCN clusters People with Concepts (tags) and Content-Objects as shown in Figure 3.8.1. This clustering is based on an ontology model (see Figure 3.8.2) and allows People (PCN users) retrieving content-objects that could have some sort of relevant knowledge.

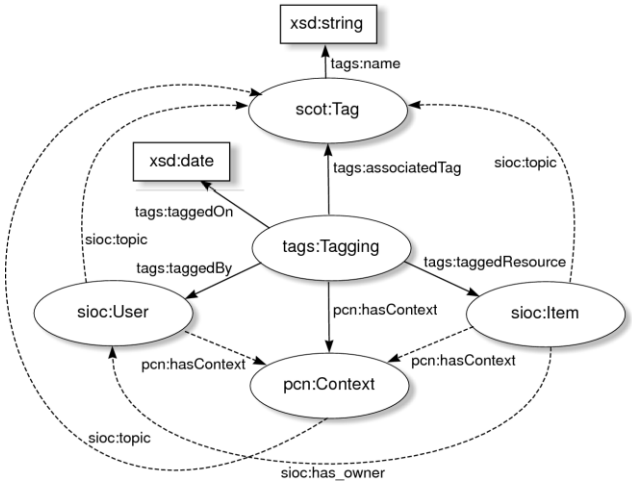


Figure 19: Simplified PCN ontology model

The PCN ontology implementation is based on a combination of existing ontology vocabularies, namely: SCOT (DERI 2008) as a skeleton conceptualization framework, SIOC (DERI 2009) for social networking information and description of the content resources, and FOAF (Brickley and Miller 2007) for personal (static profile) data, as well as a proprietary vocabulary for contextualizing other types of relationships.

In modelling the Person-Concept and Resource-Concept relationships, the PCN ontology follows the SCOT paradigm based on a tripartite 'Actor-Concept-Instance' model, proposed for aligning the semantic models with a social dimension (Mika 2005). The SCOT approach puts the concept of Tagging at the centre of the conceptualisation model. Tagging represents a single personalised act of conceptualisation and defines the person who performed it, the resource and what concepts (tags) have been used. It also can carry auxiliary information about the action, such as the time of tagging. By the context information, Tagging represents the cornerstone of the PCN ontology, effectively relating together all classes of the PCN model and providing the full set of PCN relationships, both explicit and implicit (inferred).

This leads to a kind of "dynamic" (in contrast with more "static" or declarative user profile) user profiling (see Figure 3.8.3) approach as it is based on people production rather than on a declarative profile. The main objective is to allow Knowledge Connection within People (user) communities through scanning, tagging and sharing relevant diversified and multi-format content assets within their communities.

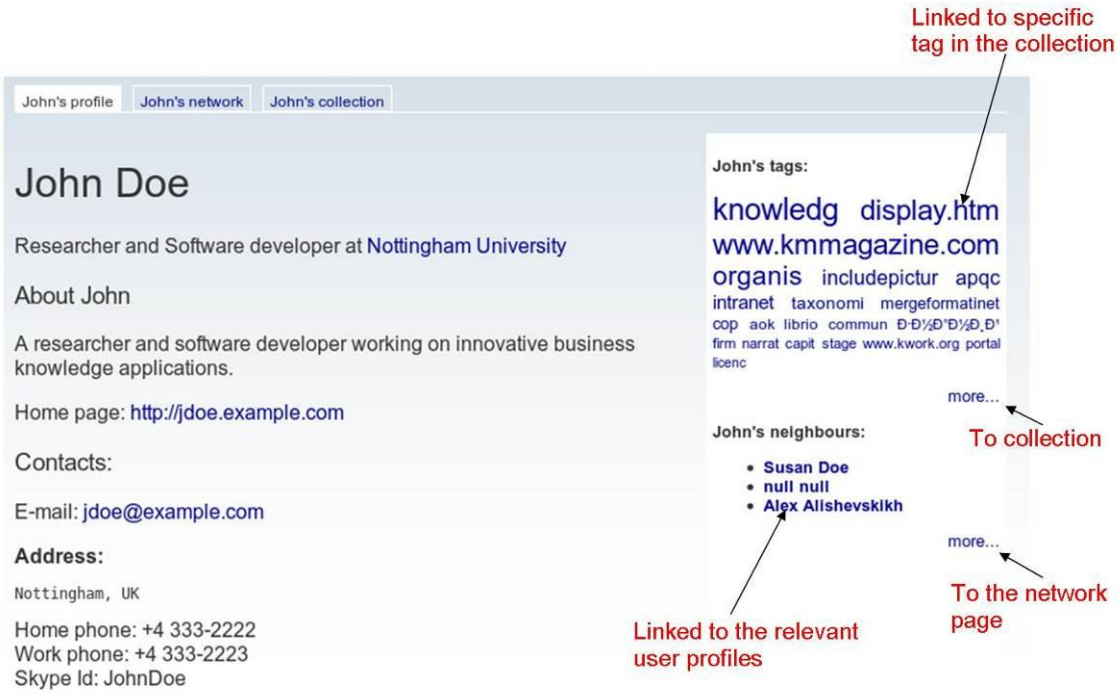


Figure 20: Example of PCN Dynamic User's Profile

PCN monitors the content locations that are duly specified by the users and incrementally update their "dynamic" profiles on a server as content objects are added, modified or deleted. The concepts in the dynamic profile can be grouped into specified "contexts", which help to organize them in accordance with different user's activities. The context provides also a social dimension of users' profiles, in representing working or social activities (e.g. projects, communities) shared by different users. The dynamic profiles of different users are aggregated into a single cluster (socio-semantic network), or network of concepts, where users are connected to each other via the shared area defined by concepts they have in common.

The PCN clustering (see Figure 3.8.4) also enables to compare People profiles (e.g. a percentage of similar concepts) and find similar interests. A future development of PCN is to compute the degree of relevance among concepts and identifies not only similarities but also possible complementarities. People are persons duly registered as PCN users.

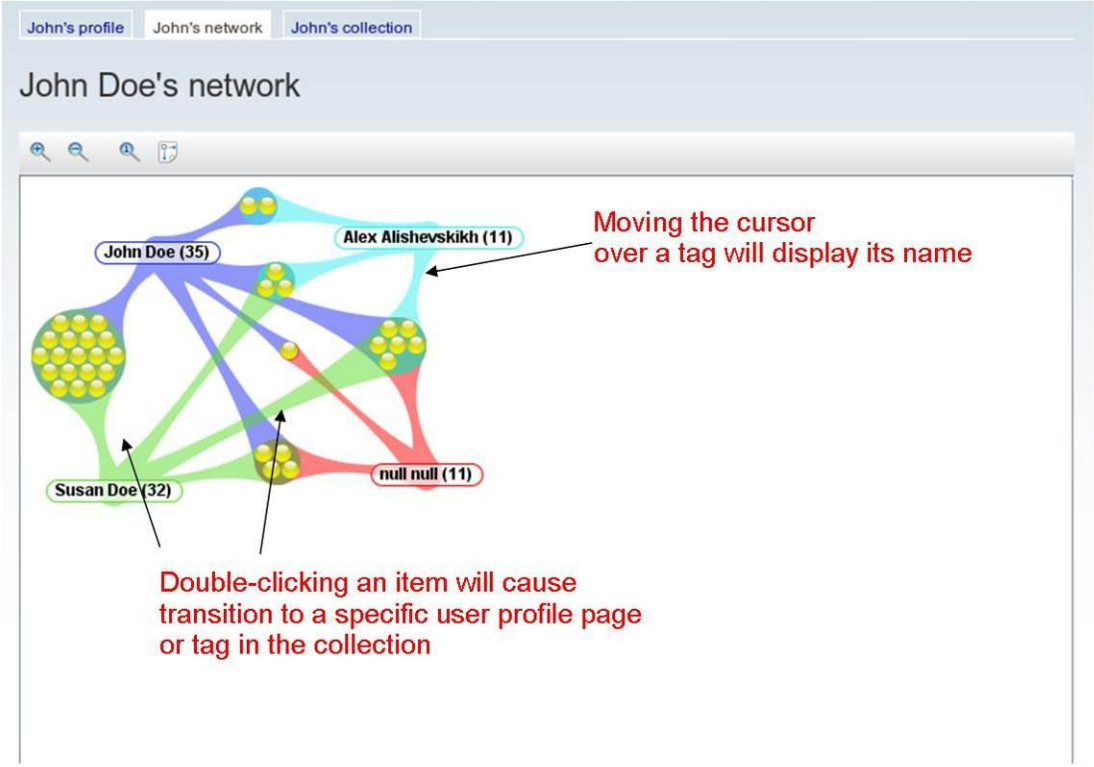


Figure 21: Example of PCN User's Network with Resources and People through Concepts

One of the main issues regarding the use of PCN was that the object location (e.g. directories, files) scanning part was standalone software that required to be downloaded and install on users' Personal Computers (PCs). Hence, a new version of PCN, named PCN2, integrates the location scanning part as services into the server and is now entering into its testing period.

3.7.1 Component Overview

Version	**PCN V2 fully distributed architecture**
Availability	**PCN 2.0 Server available on the University network**
Development	**Developed from PCN V1**
Contact Person	**Marc.pallot@nottingham.ac.uk, Kulwant.pawar@nottingham.ac.uk**

3.7.2 Installation and Technical Requirements

PCN2.0 operates as a kind of social network where users have to register for becoming members and for creating their account. Specific installation requirements are the following:

- Network-connected computer (multi-core CPU and at least 1GB of free RAM are recommended).
- Operating system, for which an implementation of Oracle Java Platform is available (Windows Server and Linux are tested and recommended).
- Pre-installed Java Development Kit 1.6 and Apache Tomcat 6.0

Host	**Platform-independent**
Application typology	**Web Application**
Application Server	**Apache Tomcat 6.0**
Major Technology	**Java Servlets, Java Server Pages (JSP), Spring MVC Framework**
Libraries	**SCAN (Smart Content Aggregation and Navigation) 2.0, Sesame 2.3.1, Lucene 3.3, Aperture 1.6, RDF2Go 4.7.3, RDFBeans 2.0**
Development Environment	**Java Development Kit (JDK) 1.6, Apache Maven 2.2, Eclipse 3.7**

3.7.3 Component description

Here below an overview of the sub-components of PCTN is provided.

3.7.3.1 PCN simplified Architecture

The PCN architecture (see Figure 3.8.4) includes the following components:

3.7.3.2 Indexing and Processing Module

Accesses document storages to retrieve full-text content and metadata (basic facts about documents and their contexts). The module contains the functional components of two types:

Crawlers: Provide access to different types of content locations, such as web-sites or BSCW server sources.

Parsers: Process documents of different types (PDF, MS Office, HTML...) to represent them in the form of token streams and RDF graphs of metadata properties. When a Crawler retrieves new document, it calls specific Parser to process it.

For every indexed document, the module produces output in the form of RDF graphs containing document metadata and token streams representing the full-text content of documents.

3.7.3.3 Storage Module

After document metadata and full-text content are retrieved, they are stored in a persistent storage facility, assembled from two components:

Document metadata repository: A high-efficient persistent graph store to keep documents metadata represented as graphs of RDF triples. Other system components can access the stored metadata with SPARQL API, provided by this graph store.

Full-text index: An inverted text index to keep the parsed token streams. The index is used for full-text search and as a source of data for text analysis functions.

3.7.3.4 Concept Extraction Module

The module integrates algorithms and techniques of statistical text analysis to identify the concepts represented in the documents content. The module retrieves the textual data from the full-text index component of the storage module and generates RDF graphs of conceptual data, represented by SCOT Tag (an extension of SKOS Concept) ontology objects.

Relationships between Tag objects and document metadata and between Tag objects and user profile graphs are the basis of PCN Logics functionality.

3.7.3.5 PCN Logics Module

PCN Logics Module is responsible for user profile management tasks, identification of conceptual similarities between profiles of different users, generation of recommendations and other logics of the PCN conceptual model.

The module integrates the **PCN data repository**: a persistent graph store to keep the data comprising the PCN Ontology Model (user profiles, concepts, contexts and resource descriptions). This ontology model is populated by importing the resource metadata from the document repository and the conceptual data provided by the Concept Extraction Module.

Stored PCN data is available via SPARQL API which is used by algorithms of this module as a source of data for its tasks.

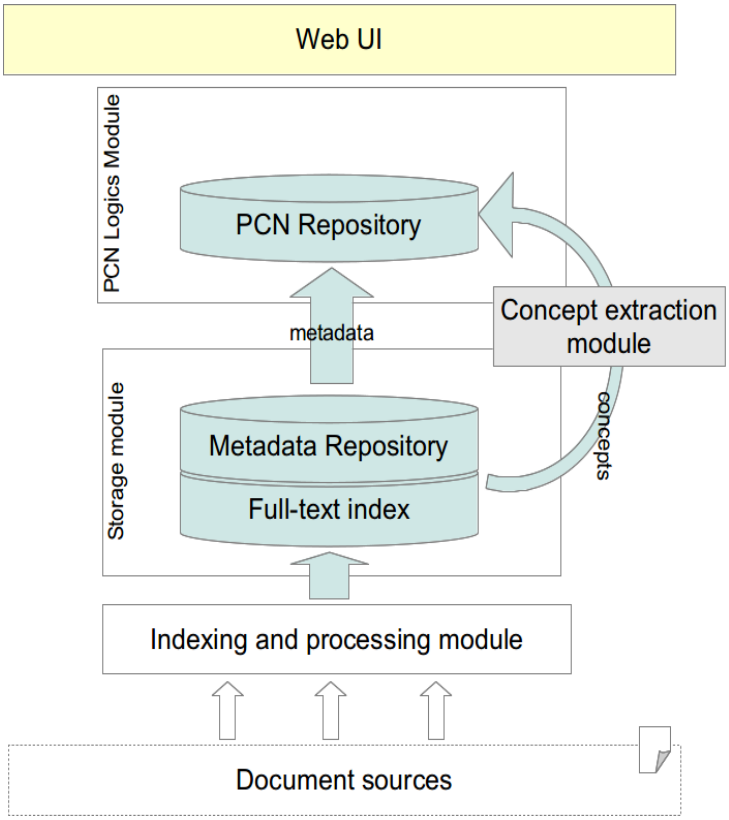


Figure 22: Simplified PCN architecture (Source: UNOTT)

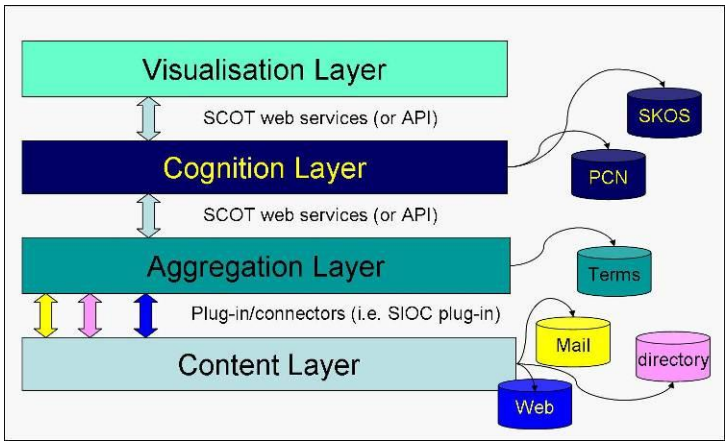


Figure 23: Simplified PCN layers (Source: UNOTT)

3.7.4 PCTN Scenario

Within the ELLIOT project, PCN is extended into PCTN ("T" means "Things") in order to identify things (e.g. sensor, actuator, device) related content-objects that are shared among a community of Living Lab stakeholders. Such a community, identified by a specific context (e.g. Living Lab), would allow stakeholders to retrieve relevant content-objects from previous experiments and re-use this experiential knowledge in their own tasks in accessing the explicit knowledge described in the content-object and implicit knowledge in talking to the owner of the content-object.

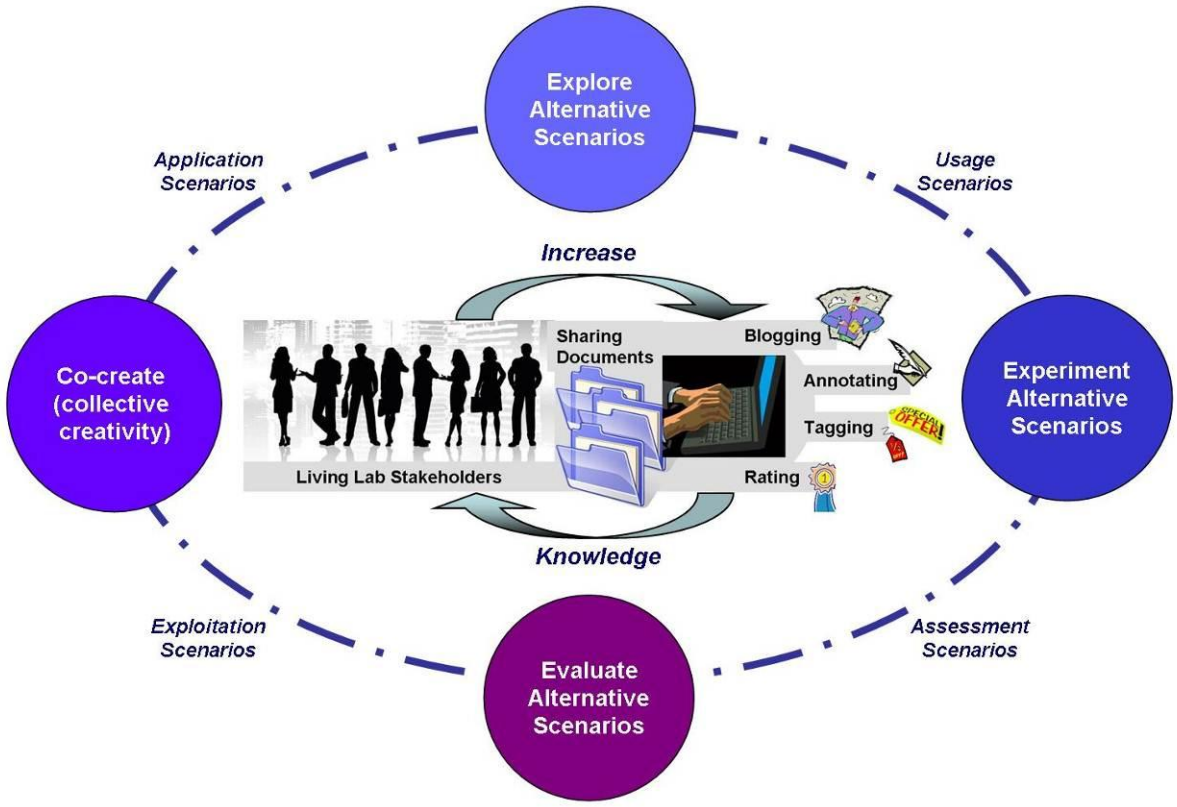


Figure 24: Produced Content Objects along the Experiential Design Process (Source: UNOTT)

It is assumed that during the LL activities and especially within the Experiential Design Process (see Figure 3.8.5) content objects are produced by LL stakeholders. These content objects could be from different types, such as weblog entries (HTML, XHTML, XML), webpages (HTML, XHTML, XML), files (e.g. MS Office types, PDF, OpenDoc), and emails (Outlook, IMAP email servers).

PCN2 extract data from location-content objects through an open-source software named "Aperture". It is a Java framework for extracting and querying full-text content and metadata from various information systems (e.g. file systems, web sites, mail boxes) through a crawler (also named parser) and the file formats (e.g. documents, images) occurring in these systems. Crawlers support the extraction of information from heterogeneous data sources.

Aperture Supported File Formats:

- Plain text
- HTML, XHTML
- XML
- PDF (Portable Document Format)
- RTF (Rich Text Format)
- Microsoft Office: Word, Excel, Powerpoint, Visio, Publisher
- Microsoft Works
- OpenOffice 1.x: Writer, Calc, Impress, Draw
- StarOffice 6.x - 7.x+: Writer, Calc, Impress, Draw
- OpenDocument (OpenOffice 2.x, StarOffice 8.x)
- Corel WordPerfect, Quattro, Presentations
- Emails (.eml files)
- ical files
- VCARD files (.vcf)
- archives (zip,tar,gz,bz2)

Aperture Crawlers Supported source types:

- File Systems (local, remote, removeable media)
- Websites and intranets
- IMAP e-mail servers
- Microsoft Outlook (alpha)
- Internet Calendar (ical) files
- mbox mailboxes
- thunderbird addressbooks
- apple addressbook

Within PCN2, a BSCW parser was developed in order to access content objects uploaded in BSCW shared workspaces.

PCN Things extension can be described in terms of the existing People-Concept-Resource-Context ontology where Things could be considered as a special class of Resource that could eventually include specific attributes like its owner, geo-localisation, type of collected data in case these information are available somewhere in a specific “Things” registry database or in scanning the content-objects.

Otherwise, in the meantime, we could simply consider “Things” as a specific Concept in the PCN ontology having a taxonomy with related terms such as “IoT”, “sensor”, “actuator”, “device” in order to identify content-objects that are “T” related resources.

References

Brickley D. and Miller L. (2007): Friend Of A Friend (FOAF) Vocabulary Specification 0.91. http://xmlns.com/foaf/spec/

DERI (2008): Social semantic Cloud Of Tags (SCOT) Ontology Specification. http://scot-project.org/scot/

DERI (2009): Semantically Interlinked Online Communities (SIOC) Core Ontology Specification. http://rdfs.org/sioc/spec/

Mika P. (2005): Ontologies are us: A unified model of social networks and semantics. In: International Semantic Web Conference, LNCS, Springer, pp. 522-536. http://portal.acm.org/citation.cfm?id=1229195

Pallot M. (2009): "The Living Lab Approach: A User Centred Open Innovation Ecosystem". Webergence Blog (http://www.cwe-projects.eu/pub/bscw.cgi/715404).

4 Conclusions and Future plan

In the scope of Task 2.3 in the ELLIOT project the interim version of the ELLIOT platform has been implemented. The fast prototype create in the scope of D2.2 and the feedback gathered form the Living Labs have been used as starting point for the implementation of this version of the software.

The platform is composed by six major items described in chapter 3:

- ELLIOT CORE GUI
- ELLIOT CORE service Layer
- ELLIOT CORE data layer
- ELLIOT Data Mining
- ELLIOT Middleware
- ELLIOT PCTN

As output from the activities of WP2 the end-users in WP4 and WP7 received the access to the online version of the platform to experiment it and the supporting material created: the user manual in annex A has been circulated and will be further improved following the platform evolution.

Next activities linked to this prototype regard the usage and feedback from end-users of the platform. The information gathered from them will be the basis for the implementation of the final version of the platform (Task 2.4) and the final version of the platform architecture (Task 2.1). The final version of the platform will be reported in D2.4 (M27) released contextually to D2.1.2 describing the final version of the architecture.

5 Annex A – ELLIOT Platform Front-End

5.1 Mock-up Overview

The ELLIOT platform GUI is composed by a web-application available on the web aiming to be the single point of access of all the operation to be done in it. The application is composed by five main areas:

- header including the project logo and the User Area where the user can see itself and log in/out
- Menu Bar to access to all the functionalities of the platform
- Workbench in which the user will interact with platform functionalities
- User Help and Guidance: a context-aware support to the user
- Platform Log: customizable log to access to the log of actions don in the ELLIOT platform

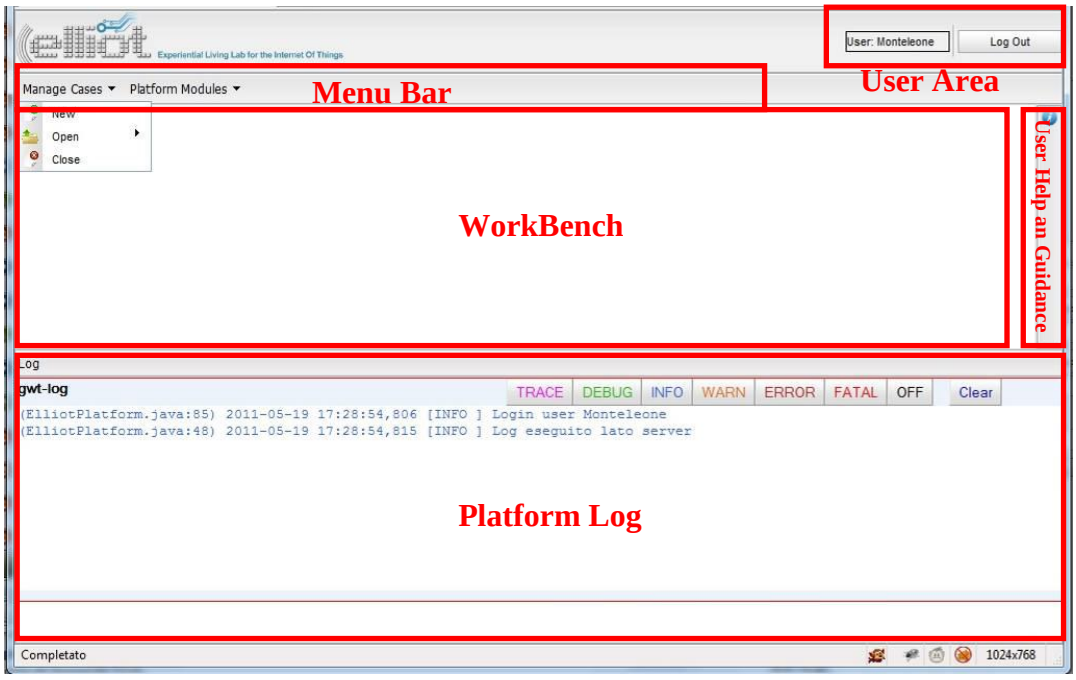


Figure 25: Overview of the ELLIOT Front End Mock-Up

At log-in time the user should specify its role (LL User or Administrator in this mock-up) and the Language (available languages: EN – DE – ITA – FR).

Figure 26: ELLIOT Platform log-in

In case the user doesn't have the credentials it has to register and the platform administrator should approve it.

First thing to do is to load the scenario using the only menu available after the login.

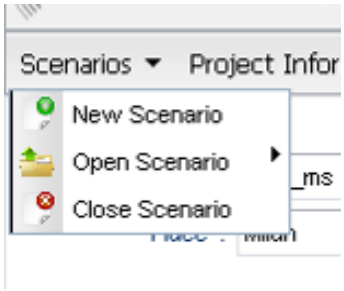


Figure 27: Initial menu

By the menu bar the platform allows the user to access to the *Scenario Overview panel* providing to the user the status of the scenario and the access to the five main modules of the Front-End, the visibility of them is depending of the role of the user.

- *Scenario Manager Module*: manages the container of the experimentation scenario – available to all
- *Co-Creation Panel:* supporting the co-creation phase including scenario goals definition, access to methodologies and tools like serious gaming (SG) and PCTN– available to all;
- *Data gathering Module*: allows the modelling of the connection of the LL to the platform and the data qualification and anonymisation supporting the experimentation phase; data pre-processing is already available in this phase supporting the calculation done on top of sensors;
- *Data analysis Module*: supporting the user in the evaluation of the observed service consumer interpreting the data gathered during the experimentation phase in terms of KSB goals – available to all;
- *Platform Manager Module*: providing the functionalities to manage users, scenarios, services, etc. – available only to the platform administrator

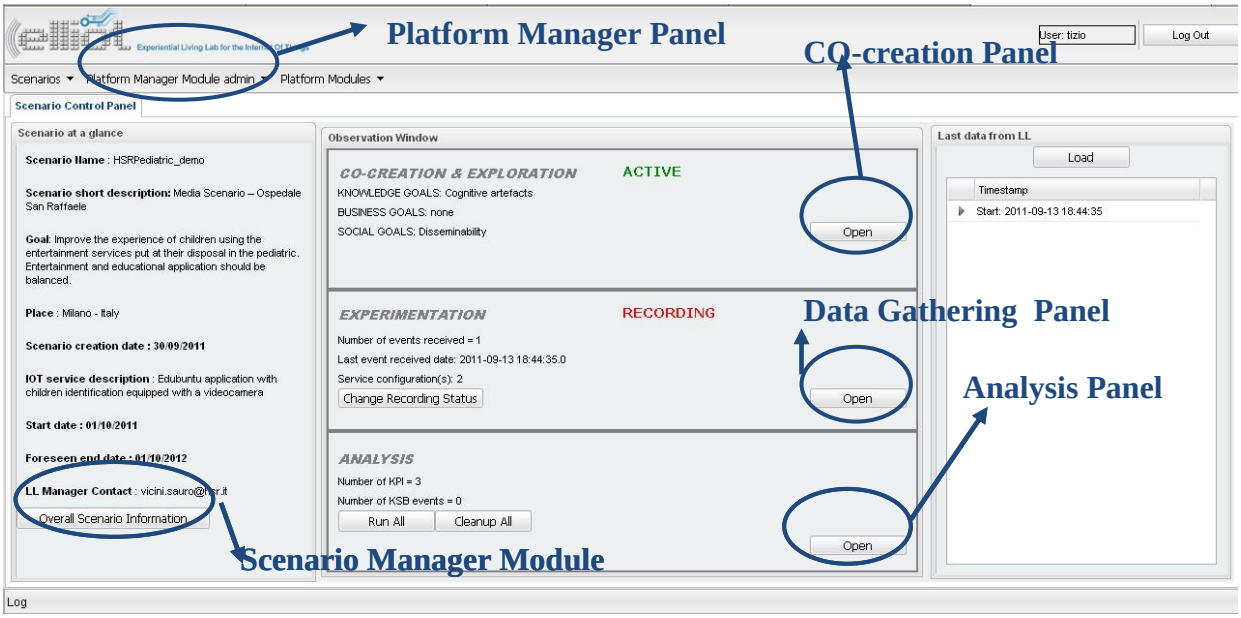


Figure 28: Overview of the access to major ELLIOT Platform modules

5.2 Scenario Manager

All scenario data can be accessed and modified in the Scenario manager. The Scenario is composed by a set of information describing it (place, date, contact) the history (how many iteration, observation time, etc) and the scenario goals in terms of KSB.

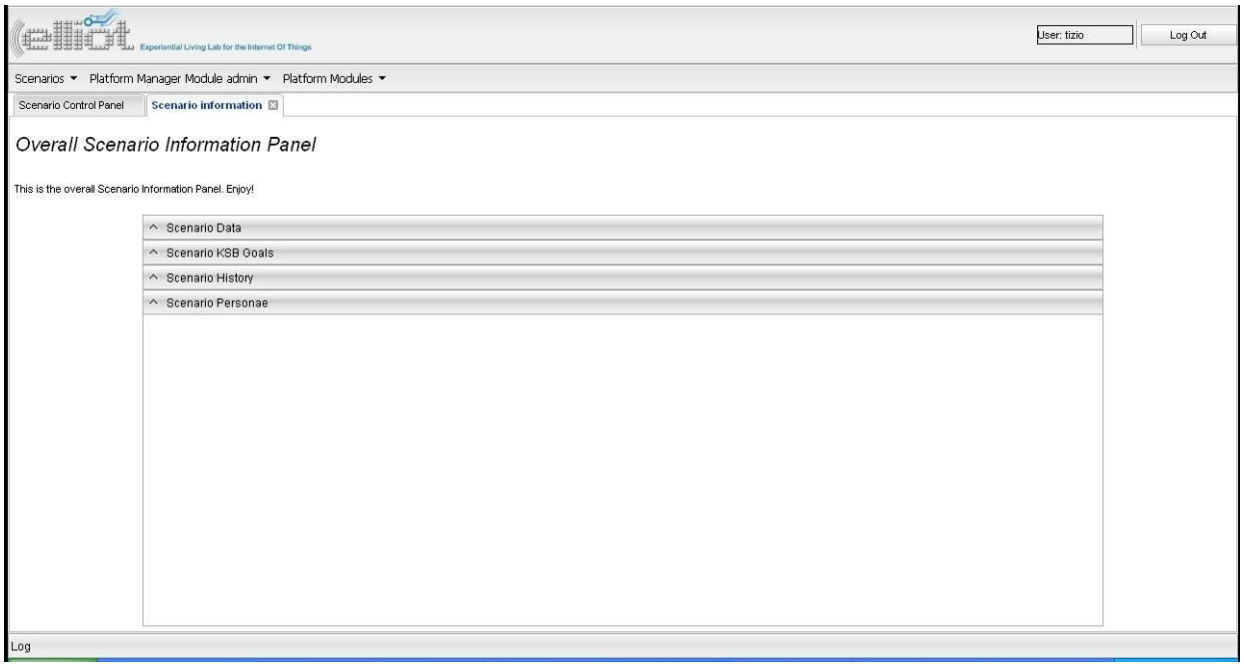


Figure 29: Scenario Manager Panel with Sections Closed

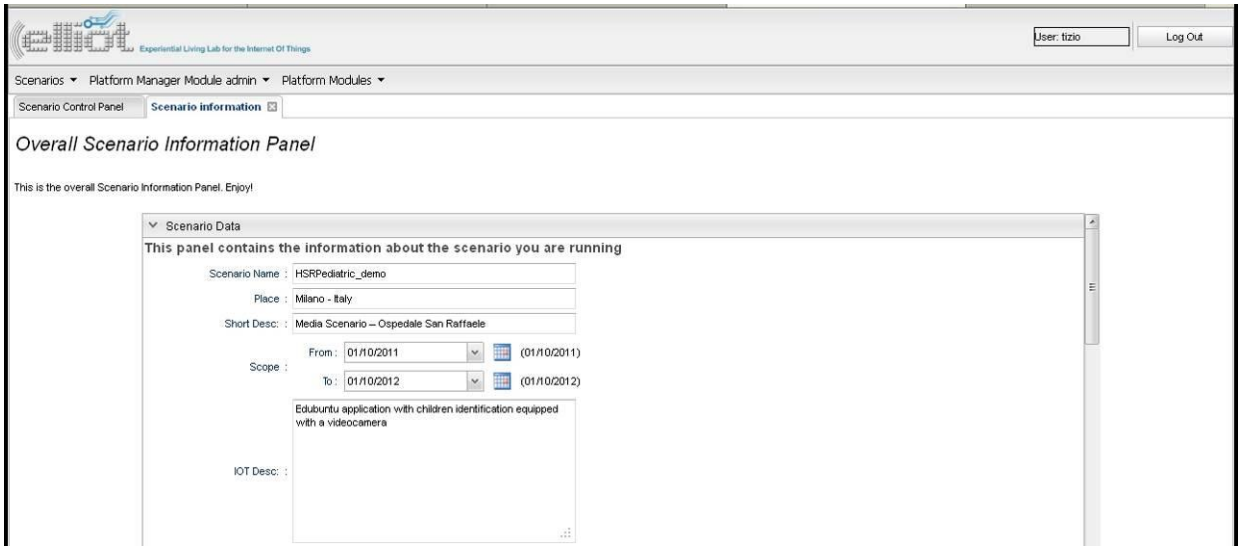


Figure 30: Scenario Manager Panel with Scenario Data Section Opened

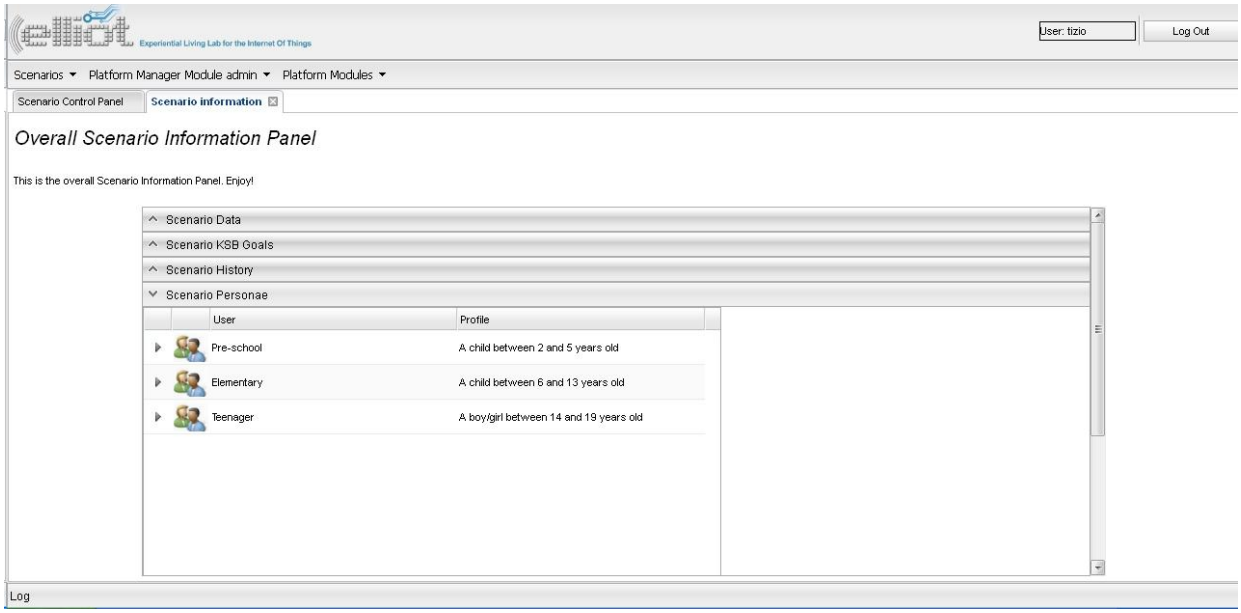


Figure 31: Scenario Manager Panel with Personae Section Opened

The user has the possibility to create a new Scenario or to end one.

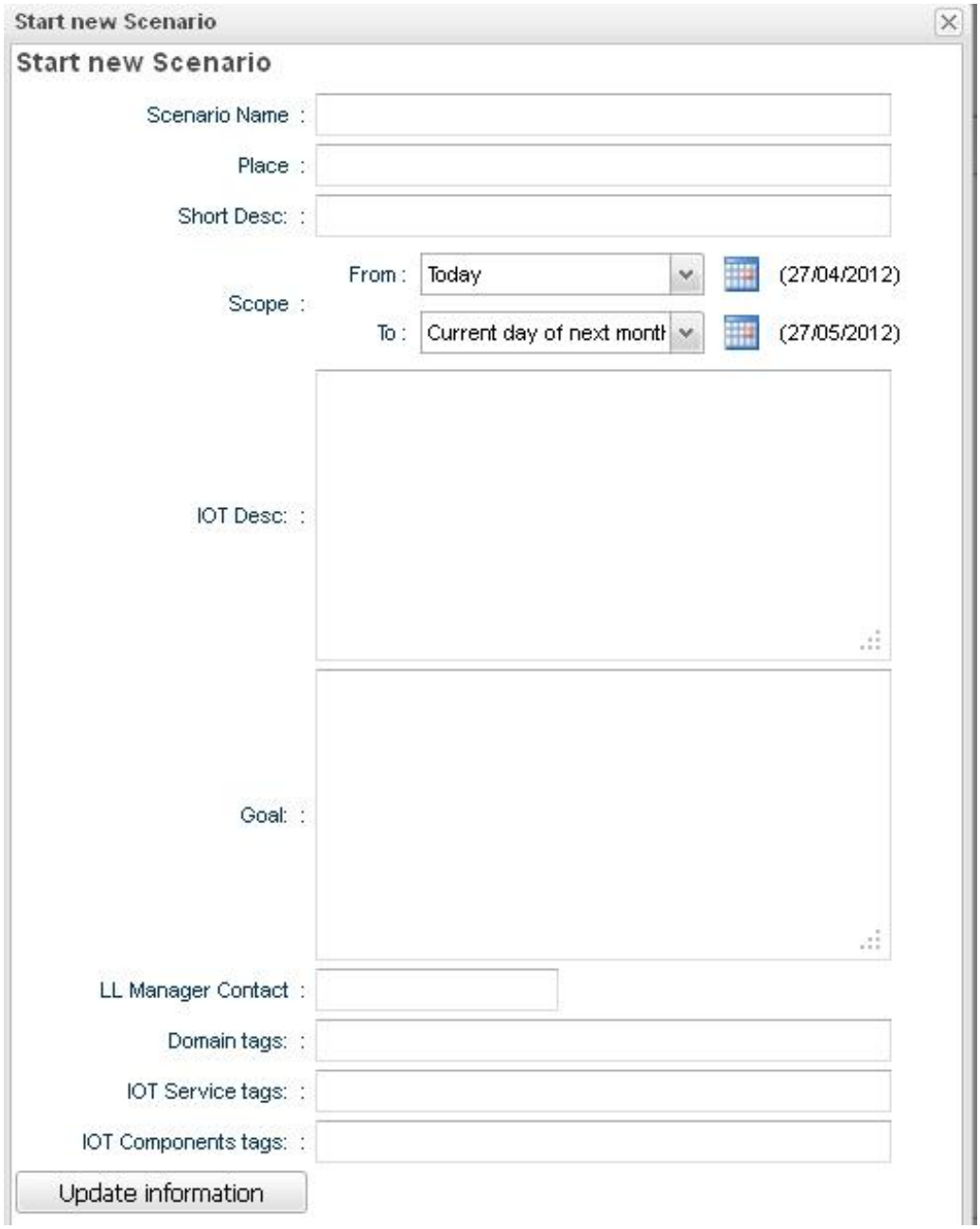


Figure 32: Start new Scenario

The Project KSB section allows the user to set the goals in KSB terms of the scenario (**Error! Reference source not found.**). On the left side the user can load the three K, S and B taxonomies implementing the generic K,S,B model and Drag & Drop them in the right side.

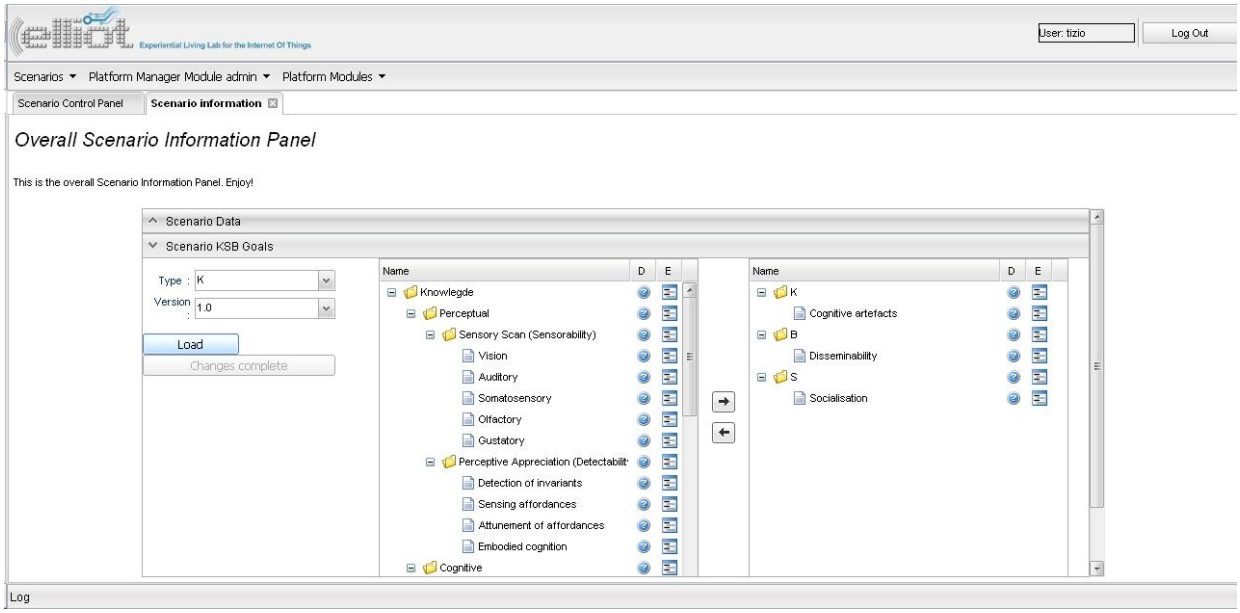


Figure 33: Selection of KSB Goals

5.3 Co-creation Module

Clicking on "*platform modules → co-creation Module"* the uses accesses to a tab by which he/her can browse all available functionalities. Opening a link the user isn't moved itself out of the tab but the system creates other tabs close to it and focuses the visualization on that.

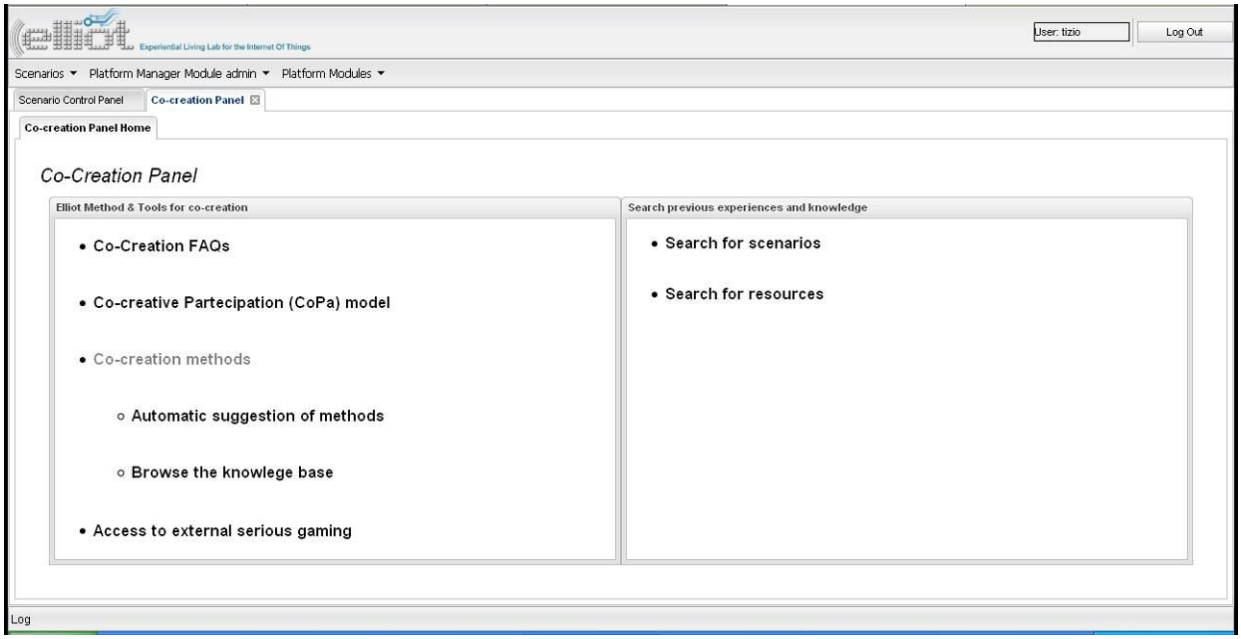


Figure 34: Co-creation Module Home Page

The page contains two sets of links:

- *ELLIOT methods and tools for co-creation* providing access to the ELLIOT knowledge base and tools for the support to people co-creation
- *Search previous experiences and knowledge* providing access to previous experiences in terms of other scenarios and/or resources like people and concepts

The "*Co-Creation FAQs*" link redirects the user on the tab containing FAQs helping the user in understanding the co-creation in respect with IOT based services.

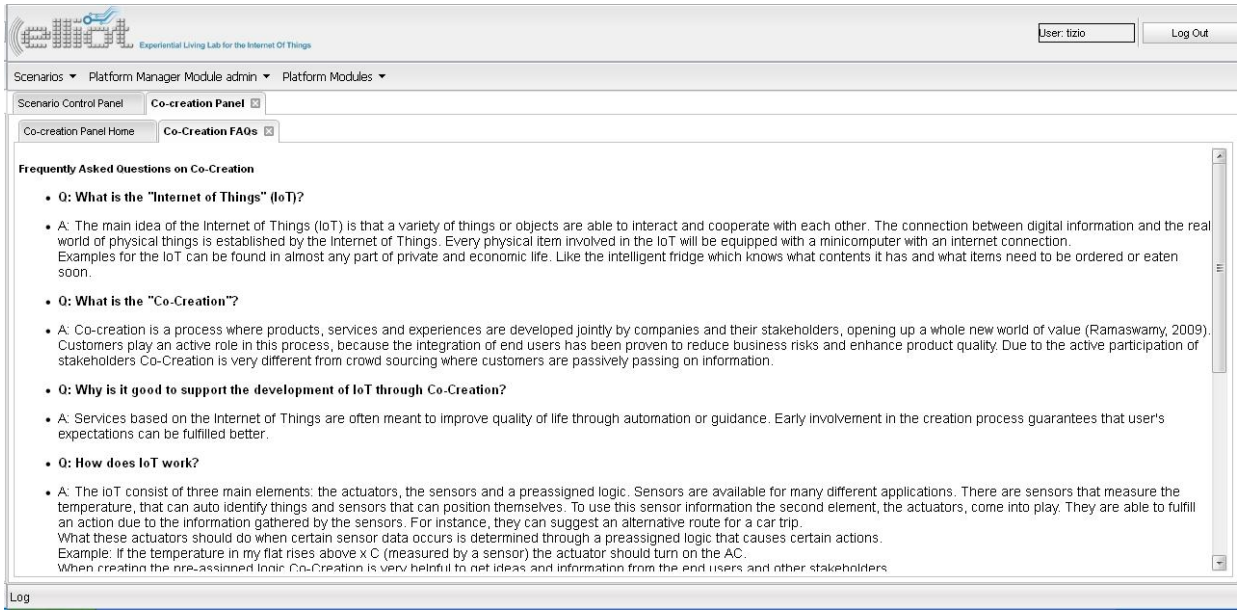


Figure 35: Co-creation FAQs

If user clicks on "co-creative participation (CoPa) model" link on the menu, the system supports the user providing a guidance of the methodology proposed by ELLIOT. By clicking on the different sections of the image there represented the user can access to the information explaining what to do in that specific phase.

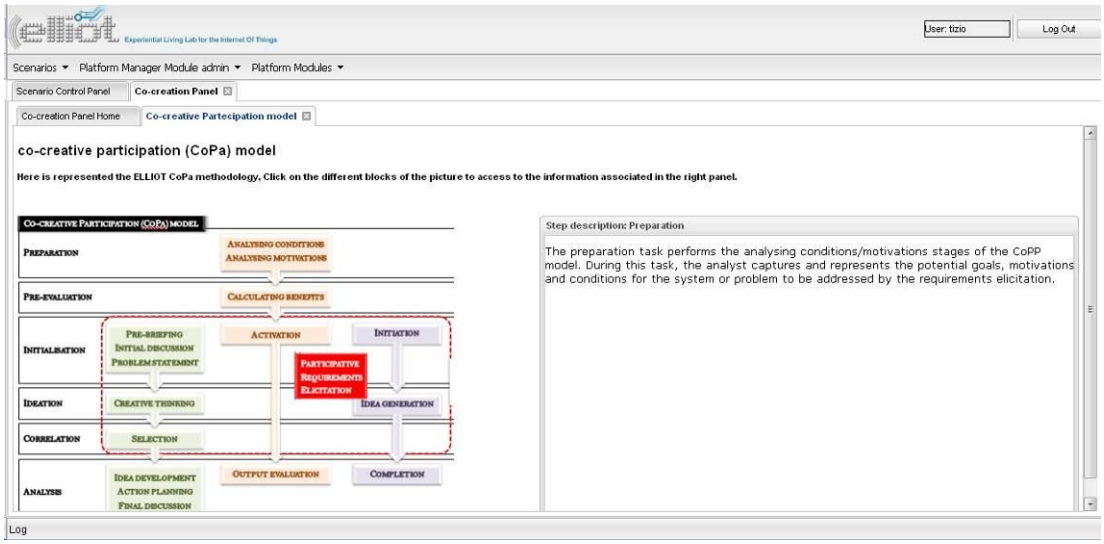


Figure 36: CoPa model tab

About the methods for co-creation promoted by the ELLIOT Platform the GUI provides two ways to access them:

- Browse the entire Knowledge Base
- Search for suggested methods

In the first case the system provides to the user the possibility to check the entire set of the methods. They are presented in form of two pictures (**Error! Reference source not found.**) to be more understandable and interesting to the user. Clicking to the boxes of the different methods the user will see the text explaining how to apply the method.

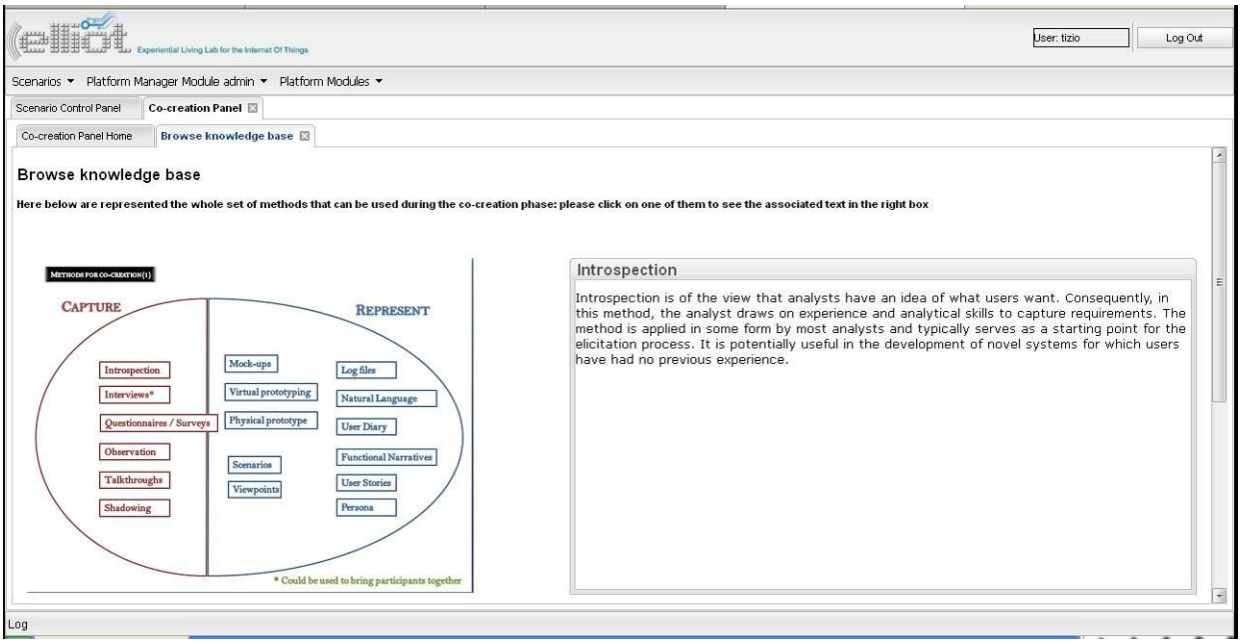


Figure 37: Browsing methods

An interesting feature of the mock-up is the possibility to have suggestions from ELLIOT about which methods apply to a specific co-creation. The *co-creation method* tab provides to the user a questionnaire (**Error! Reference source not found.**), based on the inputs the system provides back a list of suggested methods (**Error! Reference source not found.**). Clicking on them the user can see the associated description.

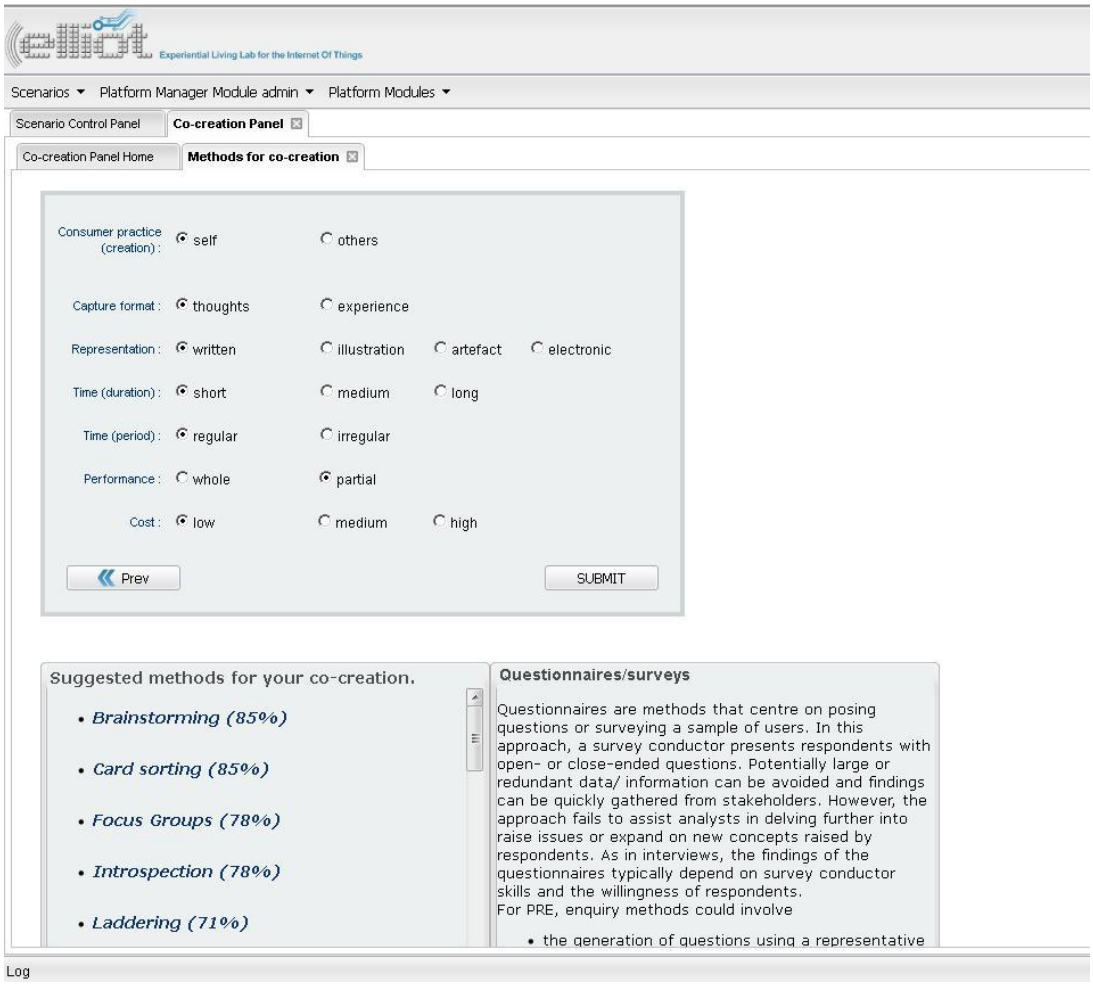


Figure 38: Fulfilled questionnaire on the co-creation to be initiated and suggested methods for the co-creation

One of the methods suggested by ELLIOT the Serious Gaming (SG); due to the fact that the ELLIOT project is focusing on them has been already provided a direct link to that tab. Anyway for the fast prototype the tab workbench is empty because the task is not yet started.

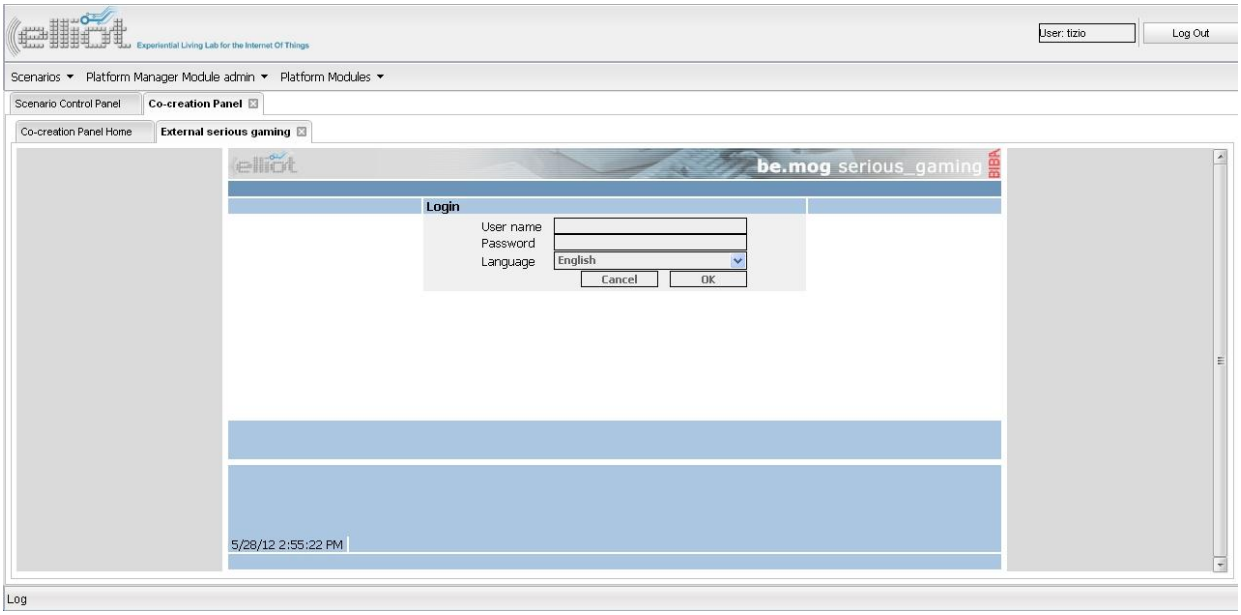


Figure 39: Serious Gaming (SG) tab

The search of past information is based on the following two panel images:

Search for Resources links the PCTN visualisation by which is possible to search for people competencies and concepts.

Figure 40: PCTN

Search for Scenarios allows searching for previous scenarios in the platform by using tags related to domain scope, IOT devices used, etc.

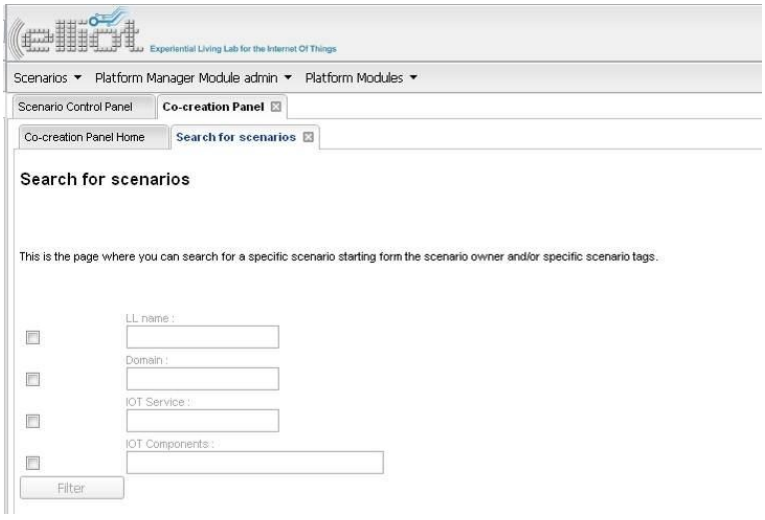


Figure 41: search for scenarios

5.4 Data gathering Module

This module allows the ingestion, qualification and anonymisation of the data collected during the experimentation phase of the LL. The content of Figure 42 is the visual menu for the user accessing the module; clicking on the picture elements user can access to the proper subsections. Half of the picture is deactivated because in charge of the Interpretation panel

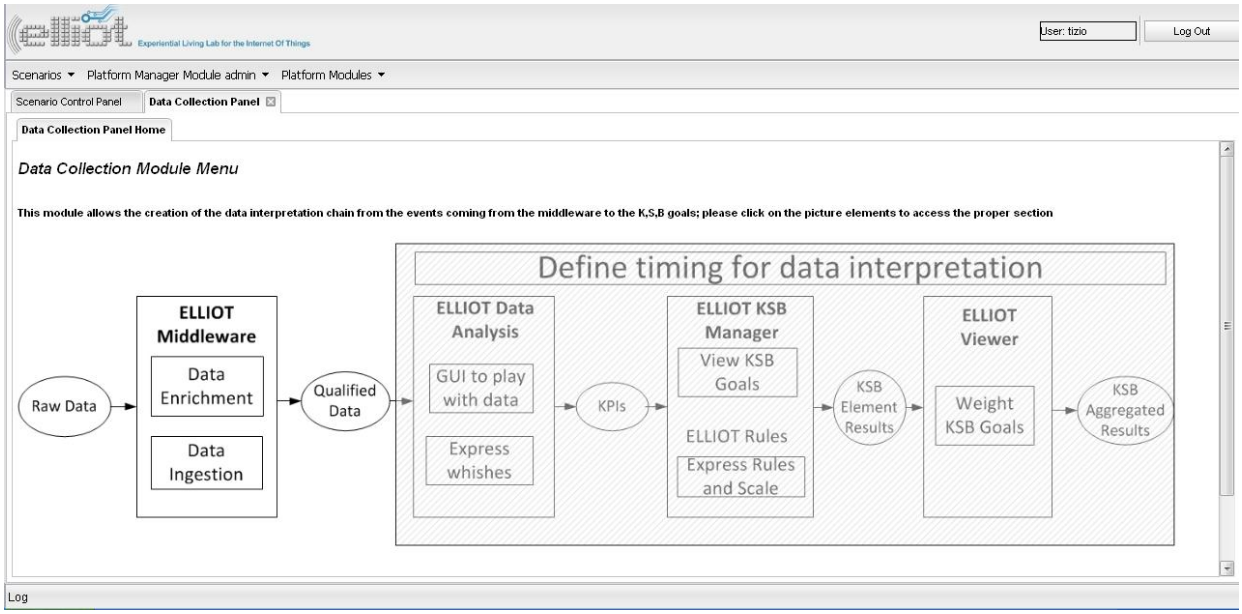


Figure 42: Data Gathering Module

In next paragraph the subsections will be addressed and explained.

Raw Data

Clicking raw data the data source structure is presented to the user to support the definition of the rules by which data will be managed and ingested.

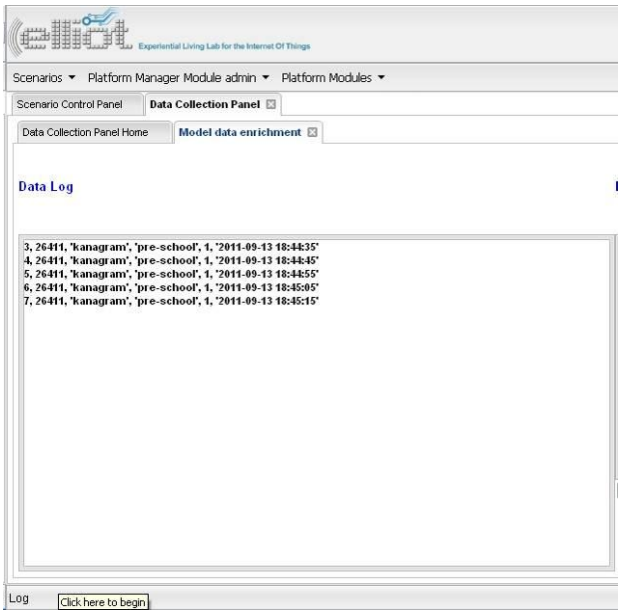
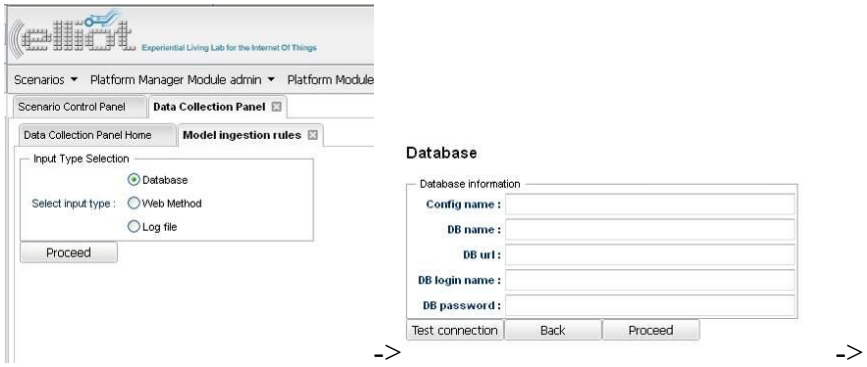


Figure 43: Raw Data

ELLIOT Middleware

Data ingestion is handled by the ELLIOT middleware composed by hydra (AKA Linksmart) and the configuration of the tool is simplified by the next interface providing a wizard for the configuration file.



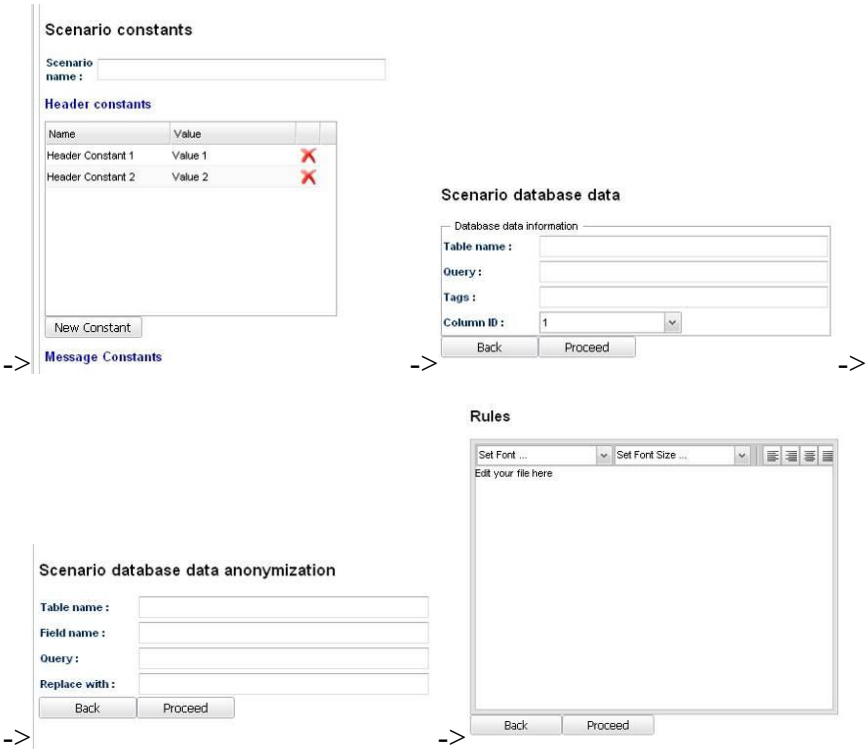


Figure 44: ELLIOT Middleware configuration

5.5 Data Interpretation Panel

This Module allows the user to set the interpretation of results setting up the functions for the data analysis and looking at results, interim results and, at any time, change the configuration of the interpretation and launching the analysis again. Next figure provides the user the access to the different components of the interpretation framework. Below the picture a description of each step is provided.

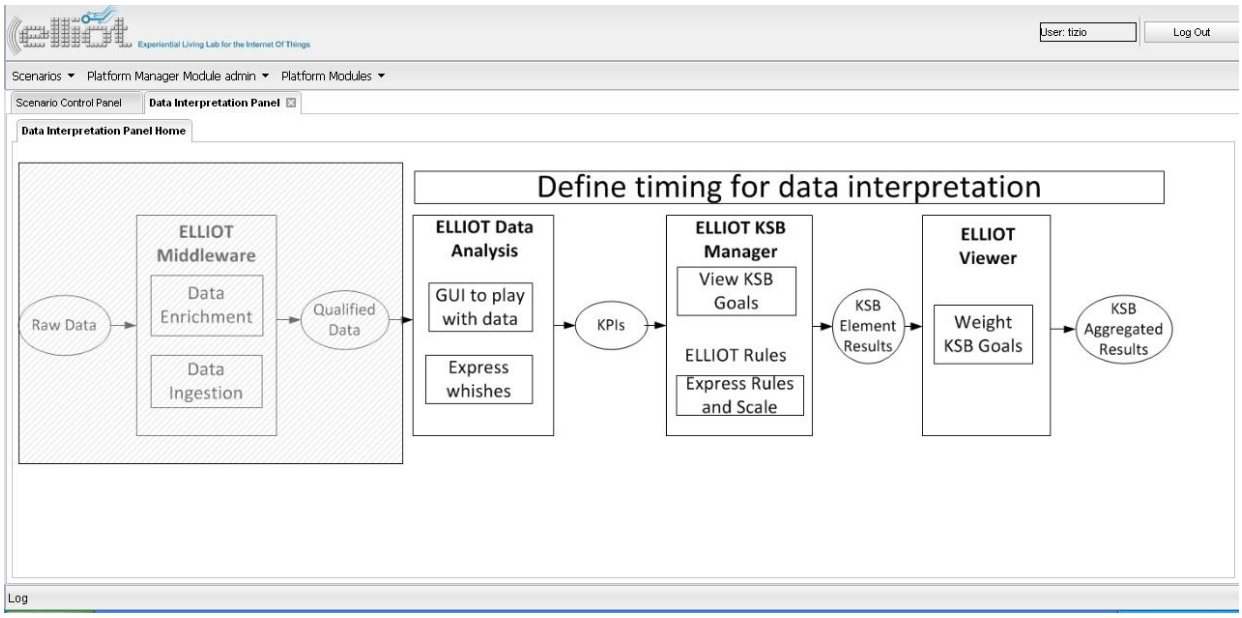


Figure 45: Data Interpretation Panel

Define timing

The first thing to do is the definition of the timing by which interpret the results: the starting fixed timing is the global one: including all data of the scenario without temporal constraints.

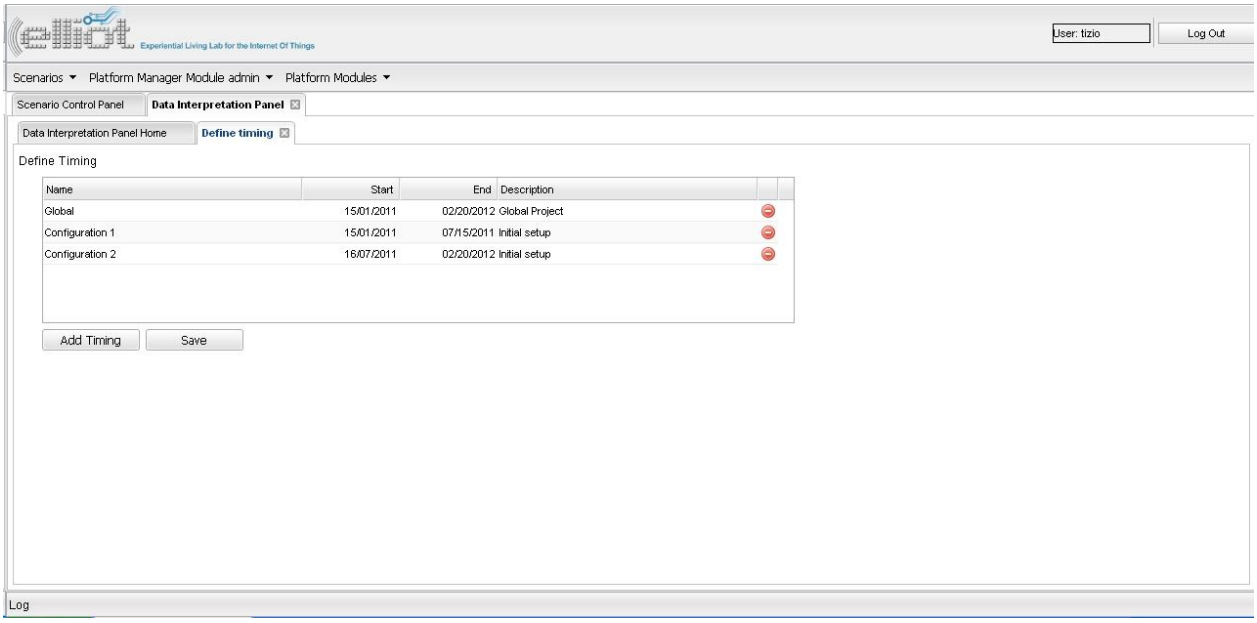


Figure 46: Timing Definition

Define indicators

Data analysis is based on indicators. Indicators are defined by the user by the following interface. In it the user can:

- Browse the ELLIOT service repository: looking at available algorithms (see chapter 3.3 for more details)
- Extract data from the database to be used in external application to understand possible indicators
- Use the GUI of Focus Lab integrated in the ELLIOT platform
- See ELLIOT Suggestions: if events coming from a device have been already analyzed in the past the Platform suggests to the user what has been done in the past.
- Model BPEL: ask to developers for a specific data analysis based on old or new web-services that will be implemented in the background in a BPEL process

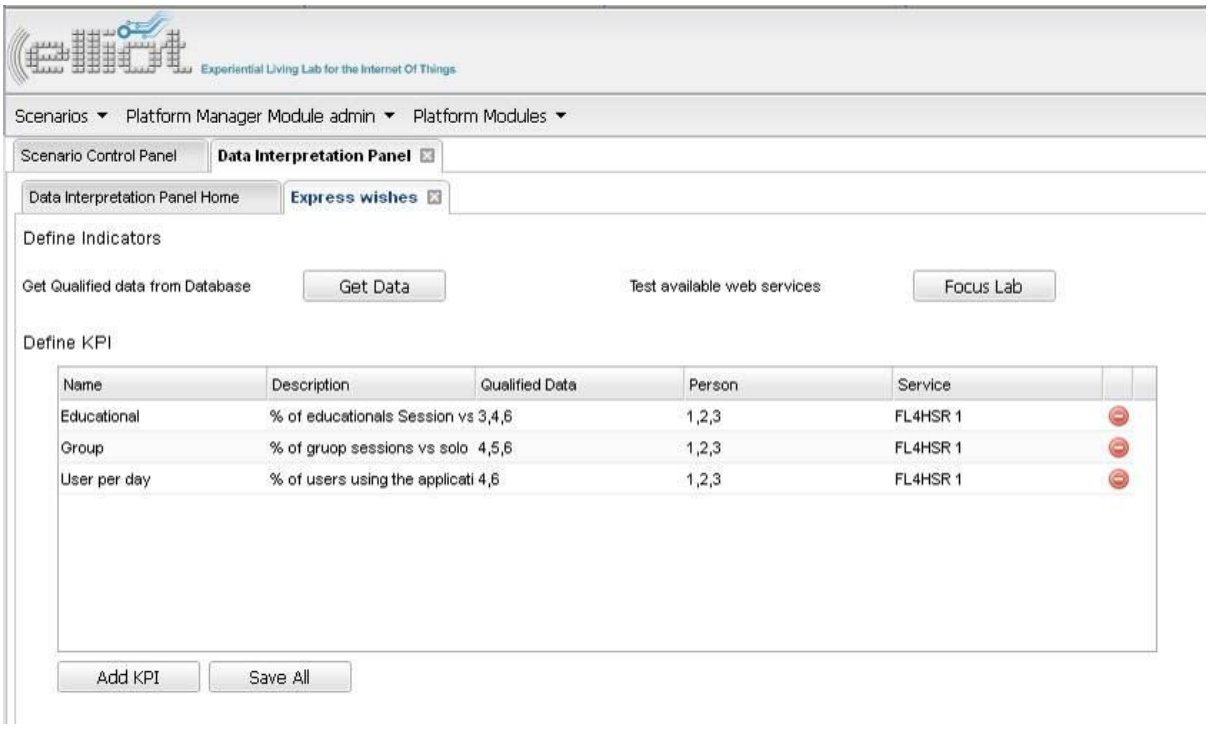


Figure 47: Define Indicators

Define Rules

The connection between indicators and KSB goals is set by the user in the following interface wrapping the ELLIOT KSB manager component. The association is based on three different levels:

- No conclusion: if for the time being there is no possibility to conclude a KPI in terms of KSB
- Simple scale (Y/N) a threeshold is set in order to reach/not reach the KSB goal
- Advanced scale (N values) several treesholds are set in order to identify the intensity of the light in case it is reached or not

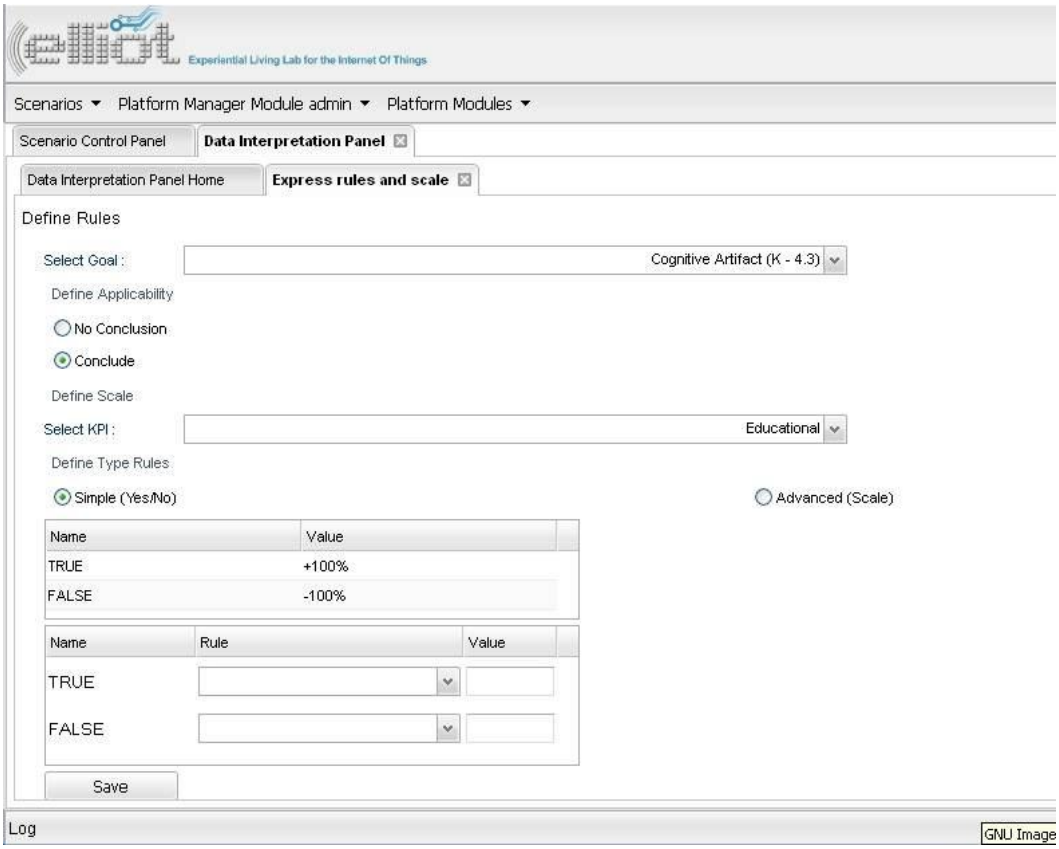


Figure 48: KPI-KSB rules

KSB results

The user access to the KSB results by the following interface; in it the user can apply filters on the results such as timing, personae and partitions. Results are displayed in different kind of graphs that the user is able to select. Of course multiple results comparing different timings are possible.

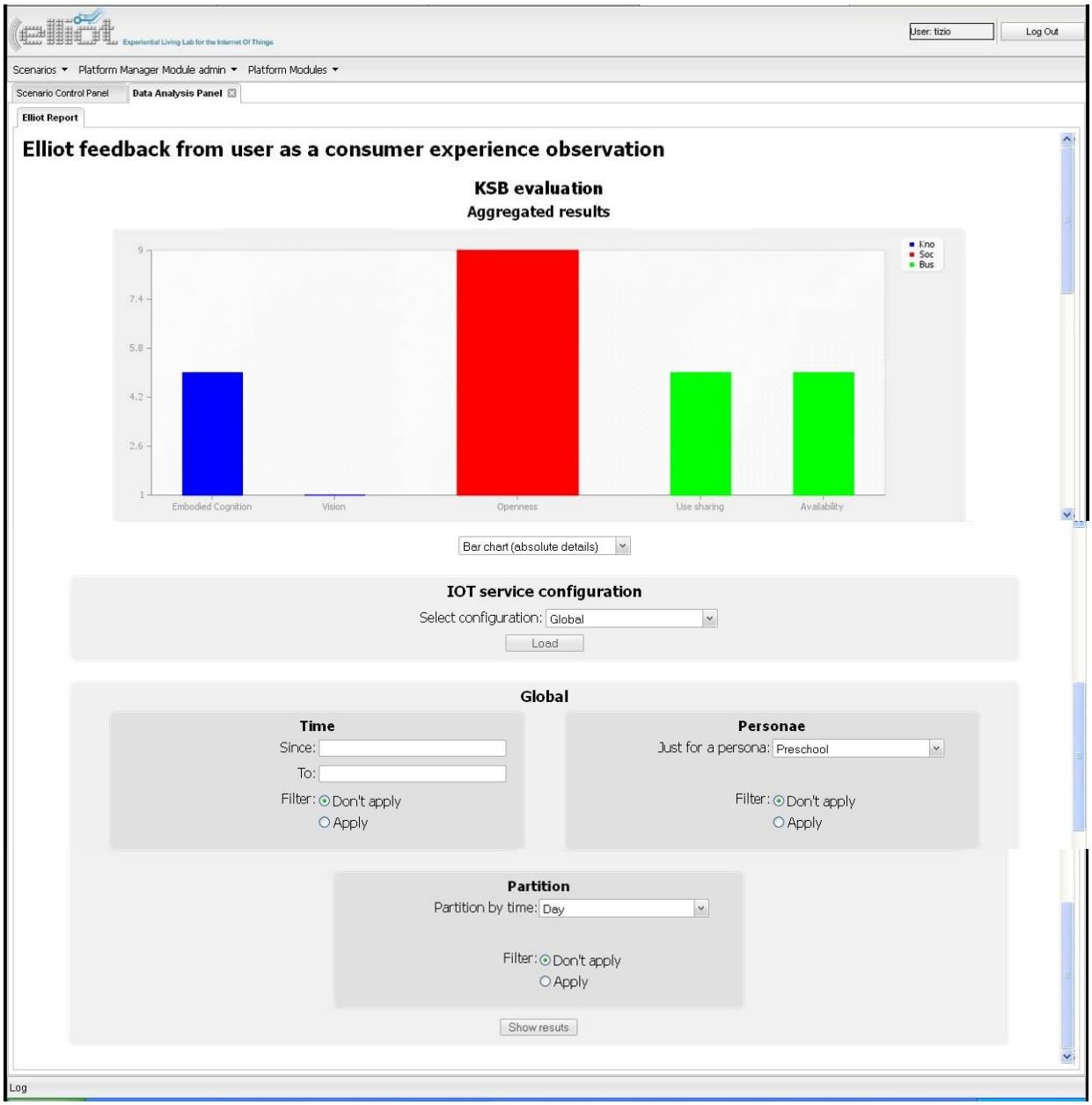


Figure 49: KSB results

Weight KSB goals

In order to summarize the results in terms of the three dimensions of the KSB model the user should weight the importance of the goals.

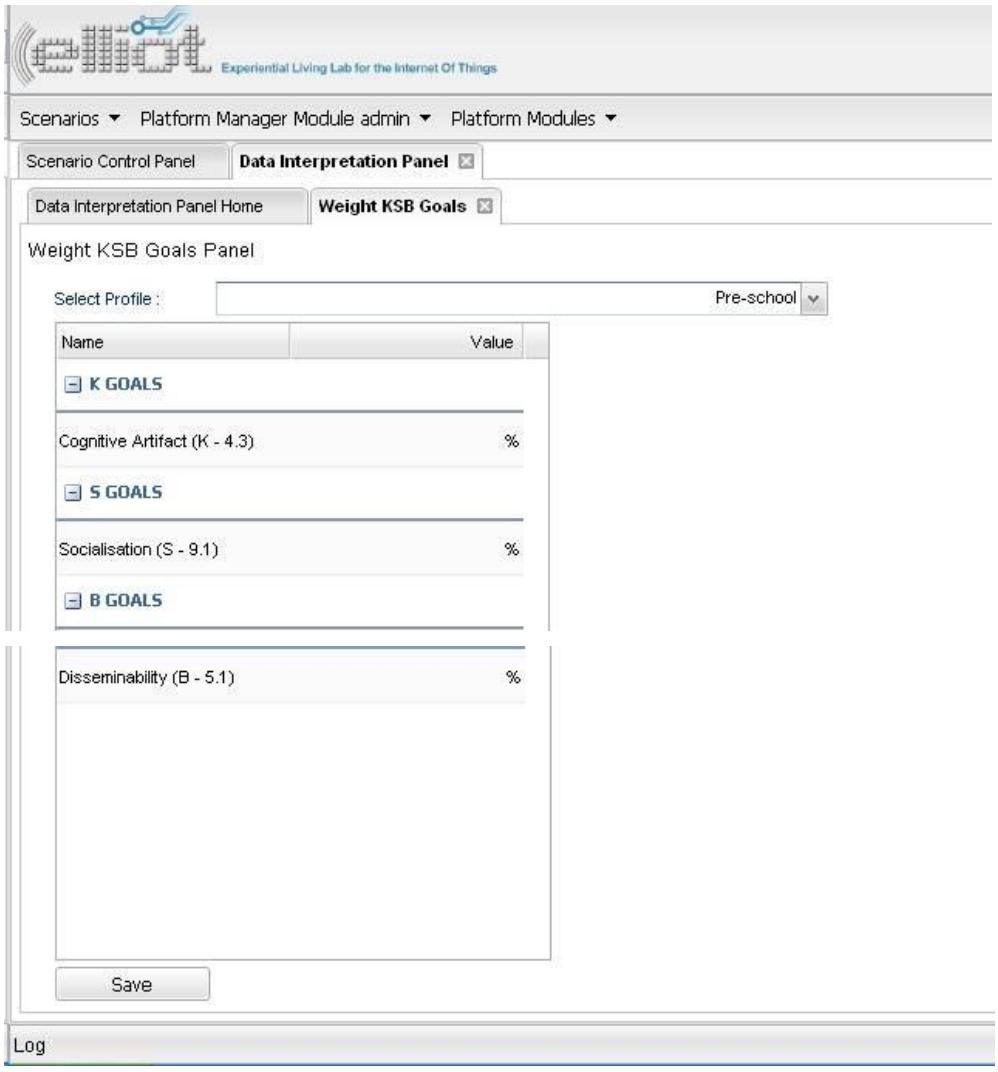


Figure 50: weight KSB goals

Aggregated KSB results

This interface provides the results of the user experience interpretation in terms of KSB assets aggregating the different KSB goals individual results. As in the case of KSB results interface the user can see the results for the global project, filter them by timing, personae, etc. Loading of more timing is allowed as well.

In the below picture is visible the global KSB view (not available in other interfaces) based on the baricentric graph.

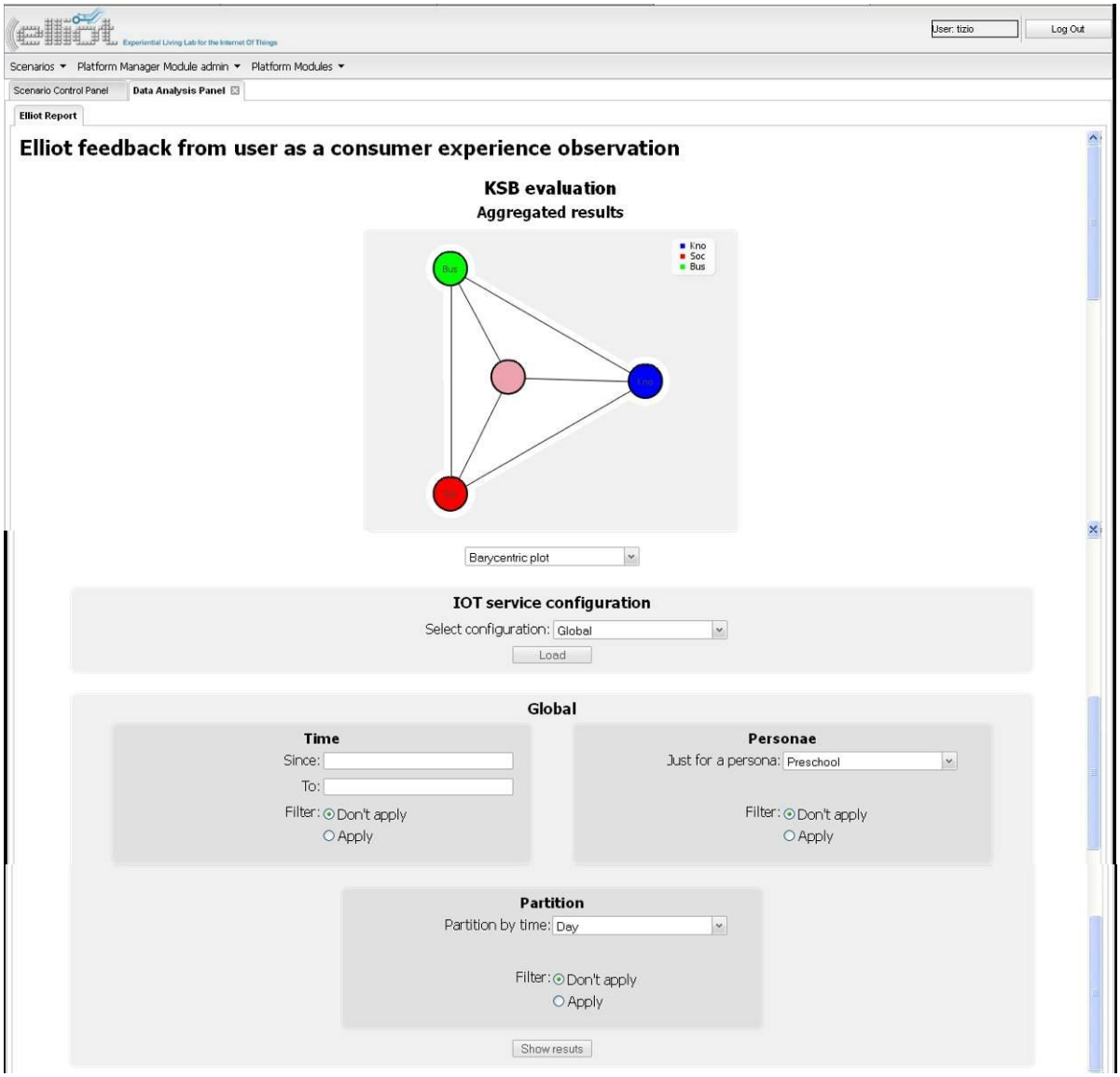


Figure 28: EAMM Module Filters

Other graphs are visible as well.

For the interpretation of results is very important to understand how they evolved along time. In next picture is visible how the user can see at multiple results comparing them.

Figure 51: multiple results visualisation

5.6 Platform Manager Module

This module, at fast prototype stage can be accessed by the administrator. Major functionalities of the module are:

- User Management
- KSB Manager

User Management

Accept user registration

At the time a user register itself to the platform the administrator should accept its registration.

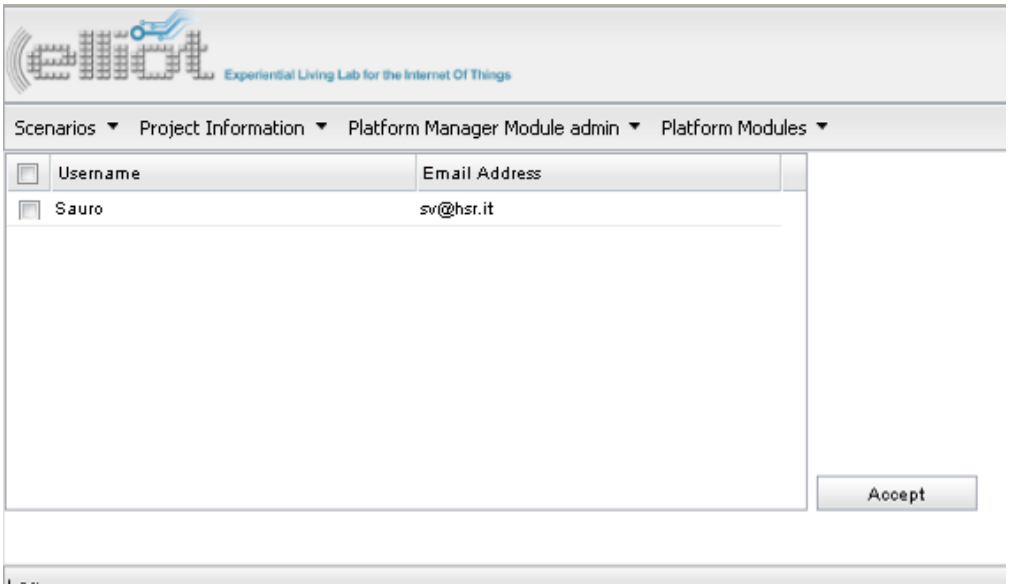


Figure 52: Accept User Registration

Add users to the project

The administrator can associate users to the different projects, In the below form by a click on the username in the left table will be available the project on the right table. Changing the associated textbox value will change accordingly user grants.

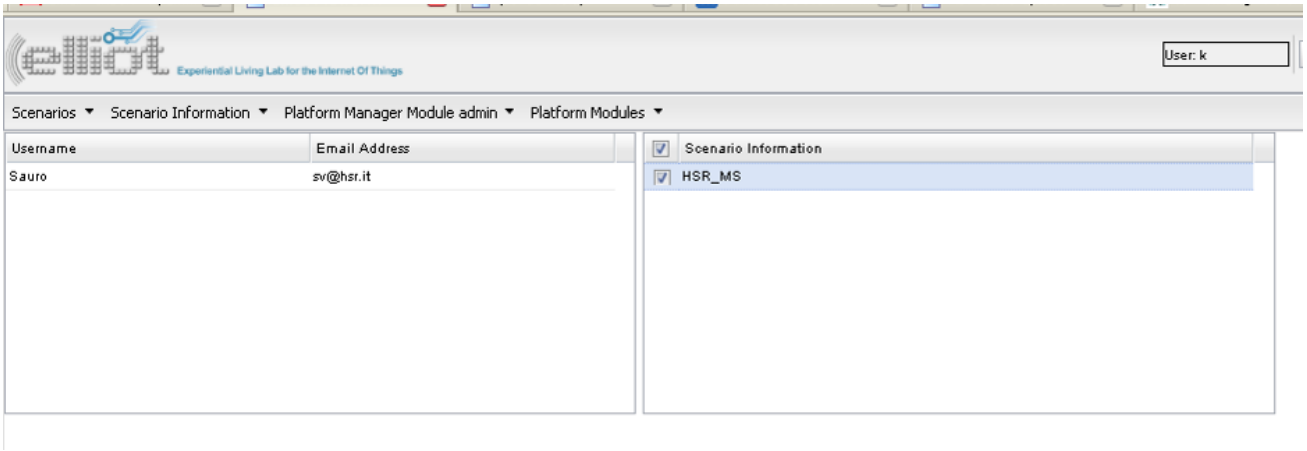


Figure 53: Add user to the project

KSB Manager

The KSB manager provides the access to the KSB model with the administration permissions; this means that the administrator can change the structure of the KSB implemented taxonomies. The Living Lab user in the ADM module can just add leaves and delete the one created. Every change to the model is saved in a new version of the taxonomy.

"Platform manager module"→"KSBmanagement"→"Edit Models"

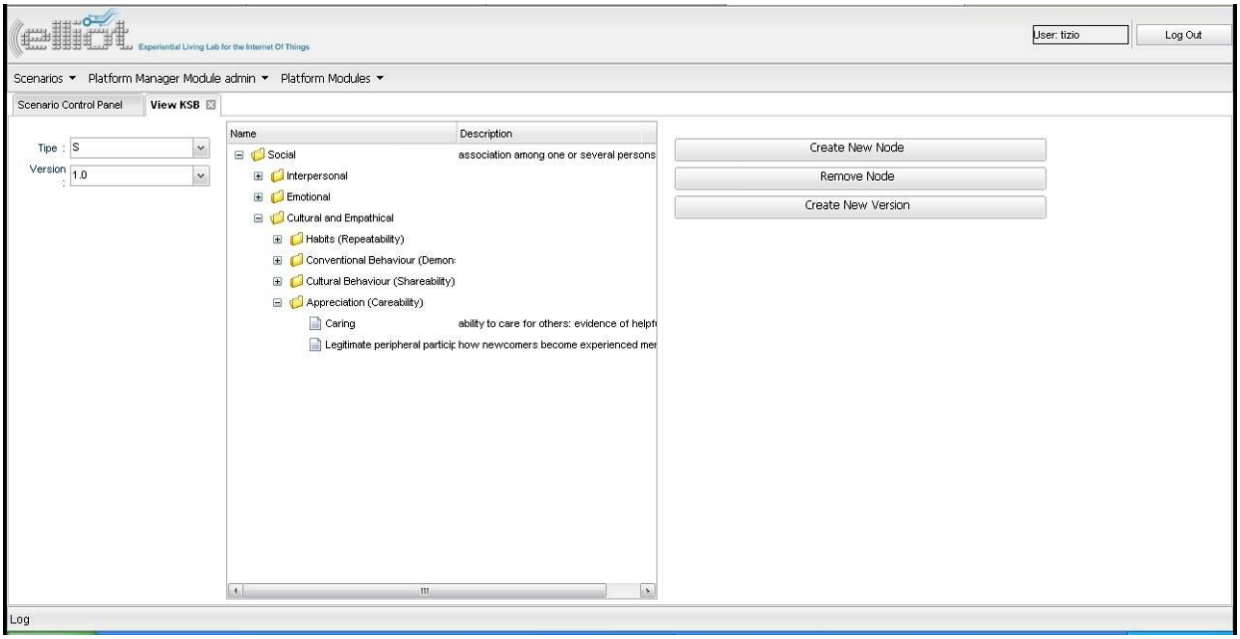


Figure 54: Extension of the K, S, B model

"Platform manager module"→"KSBmanagement"→"View Last Version"

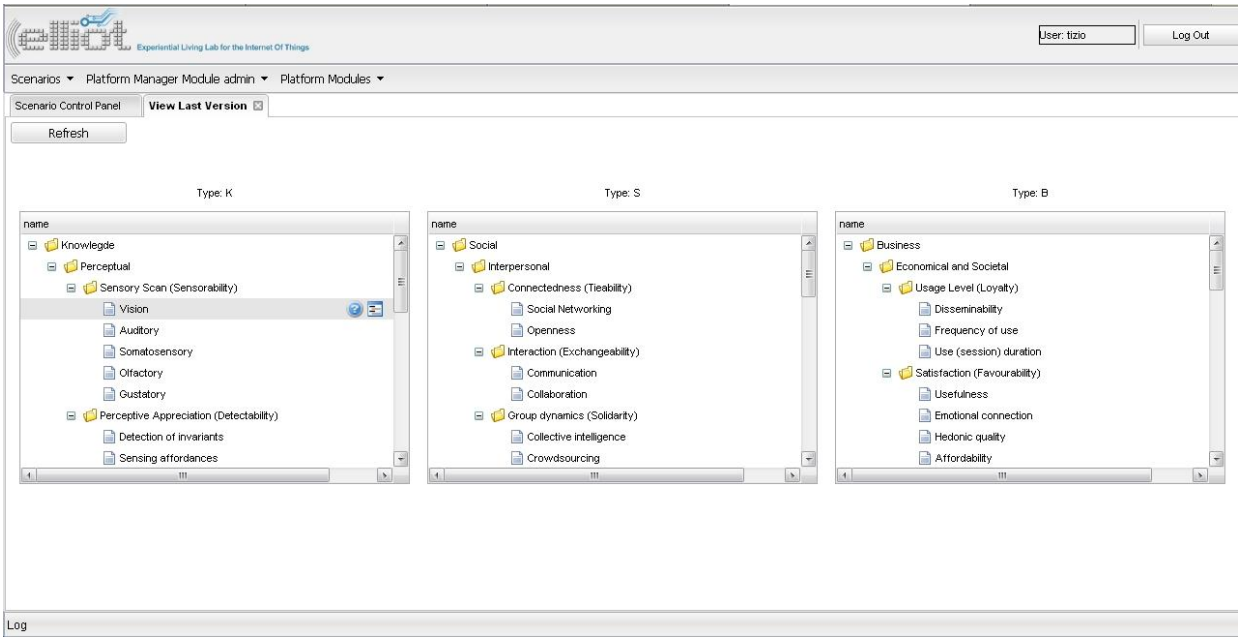


Figure 55: KSB model

5.7 User Support

The user support has been considered a key point of the platform since the beginning; in the fast prototype it is provided by three major features:

- **Multilingual support (EN – DE – IT – FRA):** all labels of the platform are provided implementing i18n support; the first languages that will be supported are the four native languages of the consortium
- **User Help & Guidance:** the platform provides a context aware guide that support the user in the understanding of platform functionalities
- **Shared Log:** shared log is visible in the bottom part of the GUI; the user can see the action did and what the other users of the platform are doing at the same time.

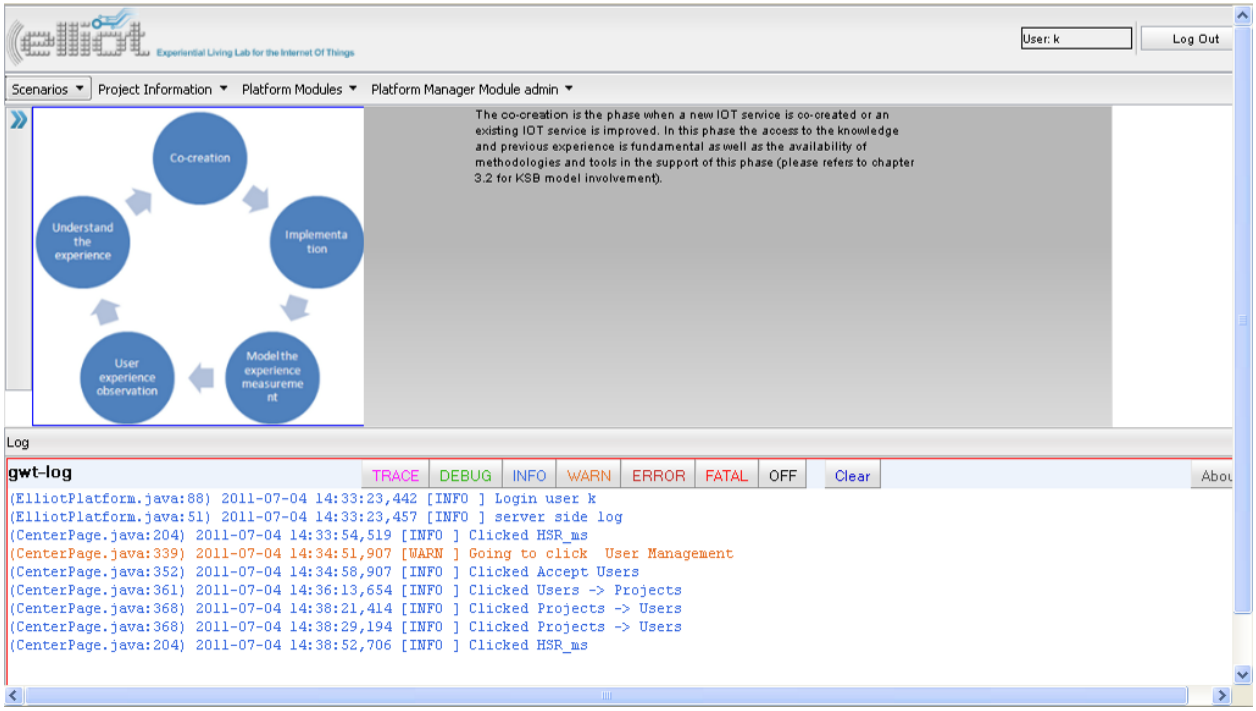


Figure 56: Context Aware Help & Guidance Support

6 Annex B – Hydra-related specifications

6.1 XML Schema for LL side data extraction specification

The XML schema given below defines the possible content for Living Lab end XML configuration files as described in Section 3.6.3.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="constant">
  <xs:sequence>
   <xs:element name="tag" type="xs:string" />
   <xs:element name="value" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:element name="config">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="dataSource">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="dbConfig" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
         <xs:all>
          <xs:element name="id" type="xs:string" />
          <xs:element name="dbName" type="xs:string" />
          <xs:element name="dbURL" type="xs:string" />
          <xs:element name="dbLogin" type="xs:string" />
          <xs:element name="dbPwd" type="xs:string" />
         </xs:all>
        </xs:complexType>
       </xs:element>
       <xs:element name="fileConfig" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
         <xs:all>
          <xs:element name="id" type="xs:string" />
          <xs:element name="absoluteFilePath" type="xs:string" />
         </xs:all>
        </xs:complexType>
       </xs:element>
       <xs:element name="deviceConfig" minOccurs="0"
        maxOccurs="unbounded">
        <xs:complexType>
         <xs:all>
          <xs:element name="id" type="xs:string" />
          <xs:element name="pid" type="xs:string" />
          <xs:element name="sid" type="xs:string" />
         </xs:all>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="scenario" minOccurs="1" maxOccurs="unbounded">
```

```
<xs:complexType>
 <xs:sequence>
  <xs:element name="id" type="xs:string" />
  <xs:element name="headerConstants" minOccurs="0"
   maxOccurs="1">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="constant" type="constant"
      minOccurs="1" maxOccurs="unbounded" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="dataDefinition">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="query">
      <xs:complexType>
       <xs:choice>
        <xs:element name="database">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="sourceRef" type="xs:string" />
           <xs:element name="tableName" type="xs:string" />
           <xs:element name="selectFields" type="xs:string" />
           <xs:element name="tags" type="xs:string" />
           <xs:element name="idCol" type="xs:string" />
           <xs:element name="isNumber" type="xs:boolean" />
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="file">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="sourceRef" type="xs:string" />
           <xs:element name="recordPattern" type="xs:string" />
           <xs:element name="selectFields" type="xs:string" />
           <xs:element name="tags" type="xs:string" />
           <xs:element name="idCol" type="xs:string" />
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="device">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="methodCall" minOccurs="1"
            maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="sourceRef" type="xs:string" />
              <xs:element name="name" type="xs:string" />
              <xs:element name="parameter" minOccurs="0"
               maxOccurs="unbounded">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="type" type="xs:string" />
                 <xs:element name="value" type="xs:anySimpleType" />
                </xs:sequence>
               </xs:complexType>
              </xs:element>
```

```
                    <xs:element name="returnType" type="xs:string" />
                   </xs:sequence>
                  </xs:complexType>
                 </xs:element>
                 <xs:element name="freqInMins" type="xs:unsignedInt" />
                 <xs:element name="tags" type="xs:string" />
                 <xs:element name="idCol" type="xs:string" />
                </xs:sequence>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
           <xs:element name="constant" type="constant"
            minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="anonimize" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="dbSourceRef" type="xs:string" />
           <xs:element name="tableName" type="xs:string" />
           <xs:element name="selectField" type="xs:string" />
           <xs:element name="whereMap" minOccurs="1"
            maxOccurs="unbounded">
            <xs:complexType>
             <xs:all>
              <xs:element name="dbColName" type="xs:string" />
              <xs:element name="valueFromTag" type="xs:string" />
              <xs:element name="isNumber" type="xs:boolean" />
             </xs:all>
            </xs:complexType>
           </xs:element>
           <xs:element name="newTag" type="xs:string"
            minOccurs="0" maxOccurs="1" />
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="rules" type="xs:string" minOccurs="0"
         maxOccurs="1" />
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
</xs:schema>
```